

PROGRAMMATION en LANGAGE ASSEMBLEUR

B. GEOFFRION

2e EDITION
revue et complétée

iAPX 186 188 et 80 386 286

EDITIONS RADIO

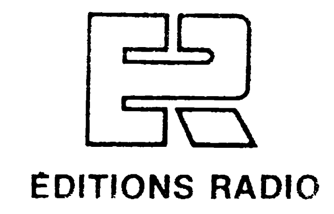
B. GEOFFRION

8086 - 8088

i APX 186 - 188 - 286 - 80386

PROGRAMMATION EN LANGAGE ASSEMBLEUR

2e EDITION
revue et complétée



ÉDITIONS RADIO
9, RUE JACOB - 75006 PARIS

Imprimé en France par Berger-Levrault, Nancy

Dépôt légal . octobre 1986
Éditeur n° 1045 · Imprimeur : 779357
I.S.B.N 2 7091 0996 4

Cet ouvrage est destiné aux utilisateurs des microprocesseurs 8086, 8088, iAPX 186, iAPX 188 et iAPX 286 (iAPX : *Intel Advanced Processor System)* désirant les programmer en langage assembleur afin de les faire travailler au mieux de leurs capacités.

Le lecteur qui n'a aucune expérience de la programmation en langage assembleur peut être découragé par la complexité de cette famille, nous lui conseillons une lecture préalable des ouvrages de A. Osborne et L.A. Leventhal, disponibles en français aux Editions Radio (Initiation aux micro-ordinateurs Niveau I et II, 6502 - Z80 - 8085 : Programmation en langage assembleur).

Cette deuxième édition « revue et corrigée » est augmentée de deux chapitres et comporte donc cinq parties :
— la première est consacrée à la structure du 8086 et aux modes d'adressage,
— la seconde détaille chaque instruction (code machine, durée...) des 5 microprocesseurs,
— la troisième rappelle les règles de la programmation en ASM 86 illustrée par quelques exemples de programmes,
— la quatrième dédiée au 80386, microprocesseur 32 bits, totalement compatible avec ses prédécesseurs, indique l'extension des modes d'adressage et les nouvelles instructions ;
— la cinquième enfin, orientée « hard », précise les consignes à respecter lors du câblage du 8086 à l'aide de la carte SDK 86 et suggère les extensions offertes par le 8087 et le 8089.

LE MATERIEL

Les microprocesseurs 8086/8088 se présentent en boîtier de 40 broches, ce qui ne révèle pas la complexité de l'organisation de la « puce ». Ils peuvent, en effet, adresser 1 M octets (M = million) ce qui exige 20 lignes pour le bus d'adresses qui est, bien entendu, multiplexé avec le bus des données (de 16 bits pour 8086, de 8 bits pour 8088).

Le schéma de la structure interne (figure 1) fait apparaître deux microprocesseurs, l'un qui exécute les instructions (EU : *Execution Unit*) avec son Unité Arithmétique et Logique (UAL) et ses registres ; l'autre qui assure la liaison avec le monde extérieur (BIU : *Bus Interface Unit*) génère les adresses grâce à un additionneur (Σ) et lit les instructions qu'il range dans une file d'attente (queue), de 6 octets pour 8086 et de 4 pour 8088.

L'unité d'exécution reçoit les instructions de la file d'attente et transmet à l'unité d'interface le résultat de son travail.

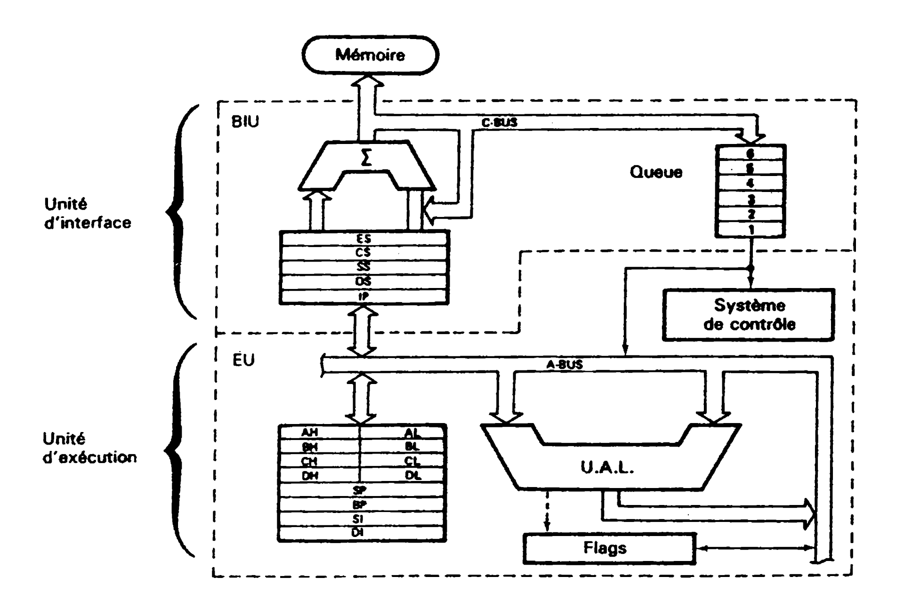


Fig. 1. — Schéma du 8086 (la queue du 8088 ne compte que 4 octets).

L'unité d'interface assure le contrôle des bus externes, lit les instructions, sans les décoder, et les range dans la file d'attente, écrit en mémoire les résultats que lui fournit l'unité d'exécution, de sorte que le bus des données est toujours occupé. Dès qu'un octet est libre dans la file d'attente, le BIU va chercher une « instruction » (fig. 2).

I. — Registres

Le 8086 dispose de 3 jeux de registres (fig. 3) de 16 bits :
— les registres généraux,
— les pointeurs et index,
— les registres de segments.

I.1. — Registres généraux

Ils sont au nombre de 4 et peuvent travailler par moitié (8 bits). Ils ont pour appellation :

— Accumulateur : AX composé de AH et AL
— Base : BX composé de BH et de BL
— Compteur (Count) : CX composé de CH et de CL

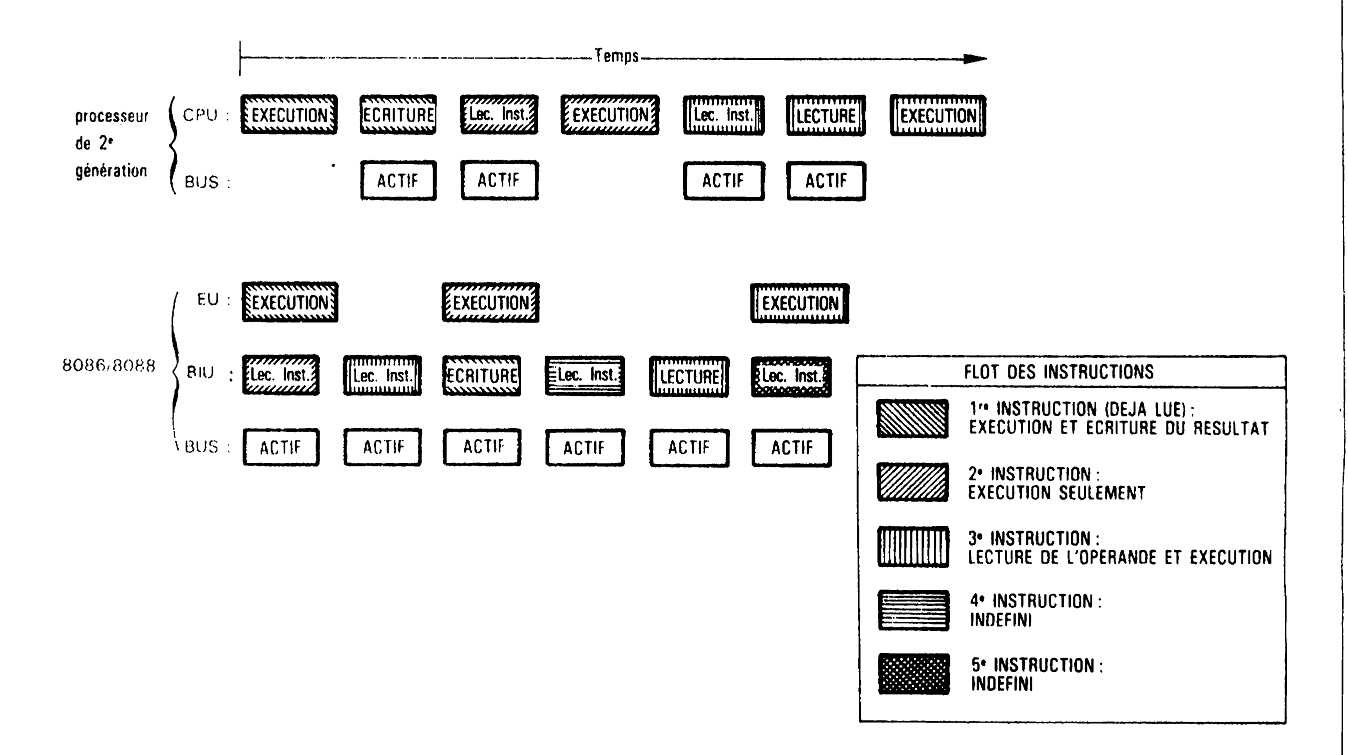


Fig. 2. — Occupation comparée du bus.

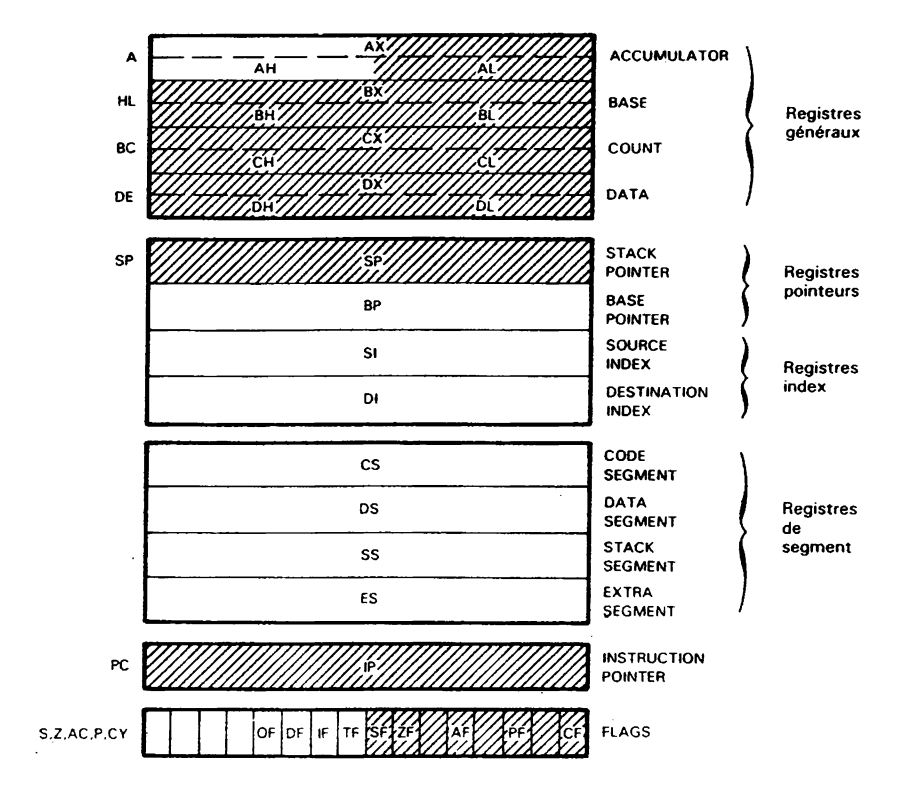


Fig. 3. — Les registres du 8086 (en hachuré, les registres du 8085).

— Donnée (Data) : DX composé de DH et de DL
(H = *High - Haut ;* L = *Low - Bas*).
En dehors de AH, ils rappellent les registres du 8085 (A, B, C, D, E, H et L).
Bien qu'ils soient « généraux », ils ont des fonctions très précises en dehors des opérations arithmétiques et logiques classiques, comme nous le verrons plus loin :
— BX sert à l'adressage en mémoire-donnée ;
— CX et CL servent de compteurs de boucles ;
— DX permet l'adressage des ports et sert, également, d'extension à AX (donnée sur 32 bits) ;

I.2. — Pointeurs et index

Parmi ces quatre registres de 16 bits, 2 sont appelés pointeurs et 2 appelés index. Les deux pointeurs SP *(Stack Pointer)* et BP *(Base Pointer)*, servent à la génération des adresses des données en particulier en pile pour SP.

Les deux index permettent la gestion de suite de mots, il s'agit de l'index de destination (DI) et de l'index de source (SI) qui peuvent également participer à la génération des adresses.

I.3. — Registres de segments

Ainsi que nous le verrons lors de l'étude de l'adressage, l'espace de 1 M octets est découpé en tranches de 64 K octets maximum, appelées « segments », référencés par rapport aux registres de segments aux fonctions précises (fig. 4).

— Segment des Codes (CS) sert à l'adressage des octets du programme (codes).
— Segment des Données (DS) sert à l'adressage des données.
— Segment de Pile (SS) gère la pile.
— Extra-Segment (ES) complète DS.

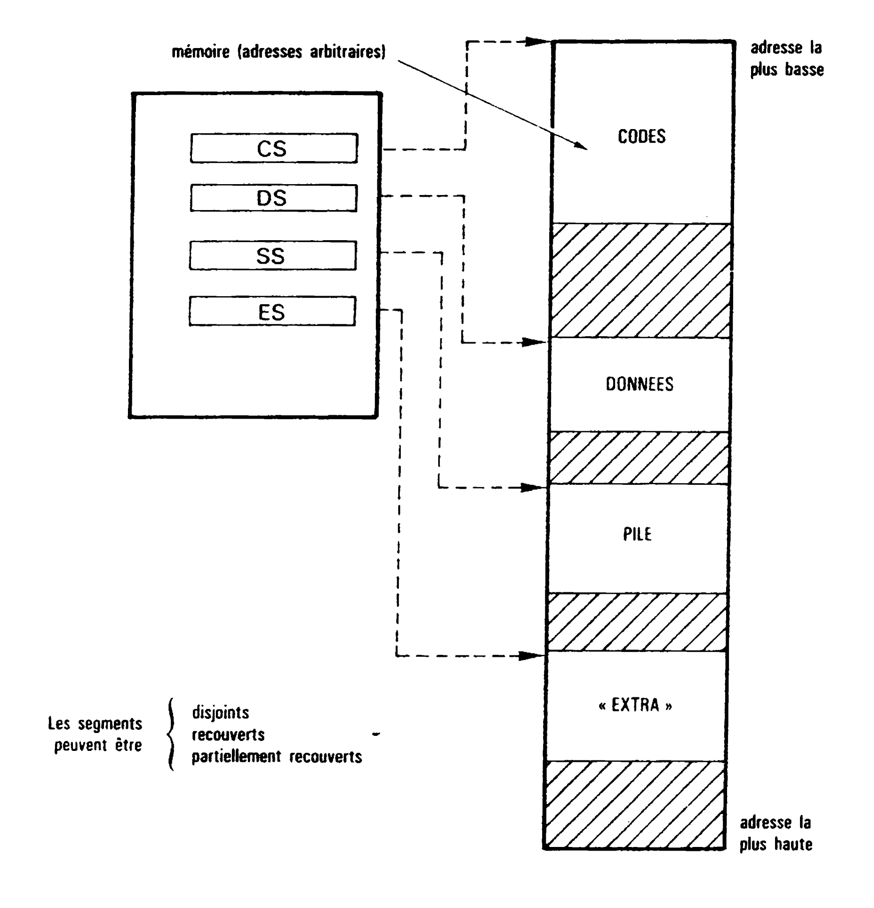


Fig. 4. — La segmentation.

I.4. — Compteur ordinal et indicateurs (flags)

Il ne faut évidemment pas oublier le compteur ordinal, de 16 bits, appelé ici pointeur d'instructions (IP) ce qui précise bien son rôle, et le registre des flags, au nombre de 9, occupant les places suivantes :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

On trouve, dans l'octet de plus faible poids, les 5 flags du 8085, à savoir :
— le carry : CF
— la parité : PF
— le zéro : ZF
— le signe : SF
— la retenue intermédiaire : AF

Le carry indique un dépassement de capacité, c'est le neuvième ou dix-septième bit du résultat.

La parité est celle du nombre de bits égaux à 1 s'il est pair PF est mis à 1.

Le zéro indique que le résultat d'une opération est nul, dans ce cas il est mis à 1.

Le signe est le bit de poids fort, huitième ou seizième, d'un nombre en arithmétique signée dans laquelle par convention : SF = 0 — le nombre est positif, SF = 1 — le nombre est négatif et écrit en complément à 2.

La retenue intermédiaire est la retenue qui se propage du quartet (4 bits, *nibble,* en anglais) de poids faible vers le quartet de poids fort.

Ces flags sont complétés par 4 autres indicateurs spécifiques :
— l'overflow, OF
— le flag de direction, DF
— l'autorisation d'interruptions externes : IF
— le pas-à-pas, ou Trap, TF, qui permet la mise au point du logiciel sans gestion externe.
Le flag de direction, DF, selon qu'il vaut 0 ou 1, permet de balayer une suite de données par valeur croissante, ou décroissante, des adresses. Il a son importance dans l'exécution des instructions du type MOVS, SCAS,...

Le flag d'overflow, OF, est nécessaire en arithmétique signée, dans ce cas le bit de plus fort poids (le huitième ou le seizième) est le bit de signe ; s'il vaut 0 le nombre est **positif,** s'il vaut 1 le nombre est **négatif** et écrit sous la forme de **son complément à 2.** Au cours d'opérations arithmétiques, il faut conserver l'information du signe, qui peut être perdue dans deux cas :
— un carry à 1 est généré sans retenue ajoutée au(x) bit(s) de signe,
— le carry a été mis à 0 mais une retenue a été ajoutée au(x) bit(s) de signe que nous analysons à l'aide de quatre exemples sur 8 bits : (H = Hexadécimal, D = Décimal).

Nous représentons, dans les opérations détaillés ci-après, par [0] ou [1] le carry et par rs[1] ou rs[0] la retenue (rs) qui s'ajoute aux bits de signe.
Le complément à 2 d'un nombre N s'obtient en remplaçant tous les 0 par des 1, et vice-versa, ce qui donne **le complément à 1**, noté $\overline{N1}$, auquel on ajoute 1.

a) 80H + 80H = (−128D) + (−128D) = −256D

	rs[0]
80H	1000 0000
+ 80H	+ 1000 0000
[1] 00H	[1] 0000 0000

Ici l'information du signe est perdue sans retenue ajoutée aux bits de signe, le flag d'overflow sera mis à 1.
Le résultat, −256D, devra s'écrire sur 16 bits FF00H, le signe étant reporté au seizième bit car :

$\overline{256D}$ = 0100H

$\overline{256_1D}$ = FEFFH

−256D = FEFFH + 1 = FF00H

b) 80H + 7FH = (−128D) + 127D = −1D

	rs[0]
80H	1000 0000
+ 7FH	+ 0111 1111
[0] FFH	[0] 1111 1111

Il n'y a pas eu de retenue, le carry est à 0, il n'y a pas perte d'information, l'overflow est mis à 0. Le résultat vaut bien : −1D.

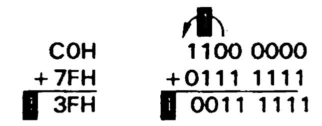
c) 7FH + 7FH = 127D + 127D = 254D

	rs[1]
7FH	0111 1111
+ 7FH	+ 0111 1111
[0] FEH	[0] 1111 1110

Ici il y a perte d'information, r_s vaut 1 et le carry est à 0, l'overflow sera mis à 1 et le résultat devra être lu sur 16 bits signés :

007FH + 007FH = 00FEH

d) C0H + 7FH = (−64D) + 127D = 63D



Le résultat est correct, 3FH = 63D, si l'on ne tient pas compte du carry, qui a été mis à 1, en raison de la retenue r_s qui vaut 1.

En conclusion, nous avons $OF = CF \oplus r_s$, et nous devons prendre des précautions si OF vaut 1. ($\oplus$ = ou exclusif, addition sans retenue).

II. — Adressage

L'espace adressable est de 1 M octets pour 8086, 8088, iAPX 186, iAPX 188 et iAPX 286 en mode « réel » ; pour ce dernier, il est de 16 M octets physiques et 1 Giga « virtuels » (1 000 M) en mode « protégé ». Ce champ d'adresses est segmenté, chaque segment peut compter 64 K octets dont la première adresse est donnée par le contenu d'un registre de segment (16 bits) multiplié par 16. Les segments peuvent être disjoints ou se recouvrir plus ou moins (fig. 4).

Les adresses, à l'intérieur d'un segment, sont :

— le contenu de IP pour le Code-Segment (segment du programme) ;

— le contenu de SP pour le Stack-Segment (segment de la pile) ;

— l'adresse effective (AE ; *EA = Effective Address*) dont nous verrons la définition plus loin, pour tous les segments.

L'adresse d'un octet, d'un mot, est obtenue selon le schéma de la figure 5.

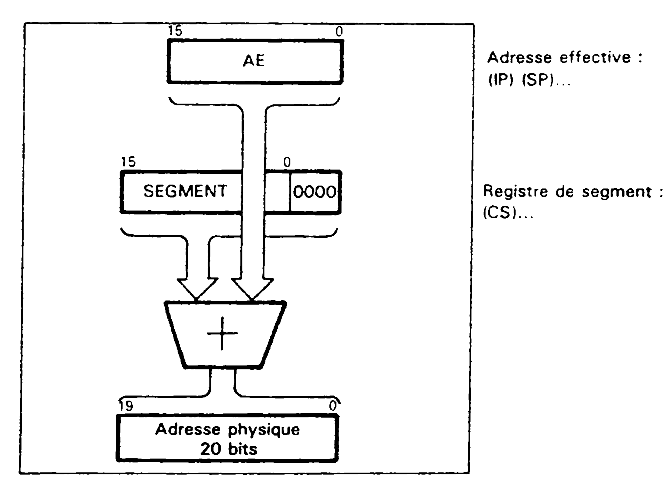


Fig. 5. — Génération d'une adresse « physique ».

Le contenu des segments peut être changé en cours de programme (sinon nous n'aurions que 4 × 64 K octets) en particulier lors de sauts ou d'appels de sous-programmes (procédures) pour le Code-Segment. Ce principe permet la « relogeabilité » des programmes moyennant certaines précautions.

II.1. — Modes d'adressage, adresse effective

Les octets d'un programme sont toujours pointés par le contenu de IP *(Instruction Pointer)* par rapport au contenu de CS *(Code-Segment).*
Les octets stockés en pile de sauvegarde sont toujours pointés par le contenu de SP *(Stack Pointer)* par rapport au contenu de SS *(Stack Segment).*

Les octets représentant des données sont pointés par une adresse effective (AE) par rapport au contenu d'un segment qui dépend de la composition de AE.
Cette adresse effective est le résultat de l'addition d'un mot (déplacement = DEP ou *DISP*) d'un registre de base (BP ou BX) et d'un registre index (SI ou DI), nous avons donc 16 possibilités pour créer une adresse de données (tableau 1).

AE (Adresse Effective)	Par rapport au segment
DEP	DS
(BP)	interdit
(BX)	DS
(SI)	DS
(DI)	DS
(BP) + DEP	SS
(BX) + DEP	DS
(SI) + DEP	DS
(DI) + DEP	DS
(BP) + (SI)	SS
(BP) + (DI)	SS
(BX) + (SI)	DS
(BX) + (DI)	DS
(BP) + (SI) + DEP	SS
(BP) + (DI) + DEP	SS
(BX) + (SI) + DEP	DS
(BX) + (DI) + DEP	DS

Tableau 1. — Génération de l'Adresse Effective et segment de référence.

Sans oublier que DEP peut être de 16 bits, ou 8 bits (voir codage des instructions).
Ces adresses, référencées par rapport à des contenus de registres de segments bien précis manquent de souplesse, aussi pouvons-nous en cours de programmation préciser qu'une donnée est adressée par rapport au contenu d'un autre segment, CS, par exemple, grâce à un « préfixe » en code machine *(override prefix)* découlant, en assembleur de l'indication du segment choisi qui peut être CS ou SS à la place de DS *sauf* pour l'index de don-

née (DI) dans le cas de traitement de suite de données référencée par rapport à ES (Extra Segment). Dans le cas de l'emploi du contenu de BP pour constituer une adresse effective, avec ou sans registre d'index, le segment de référence est SS.

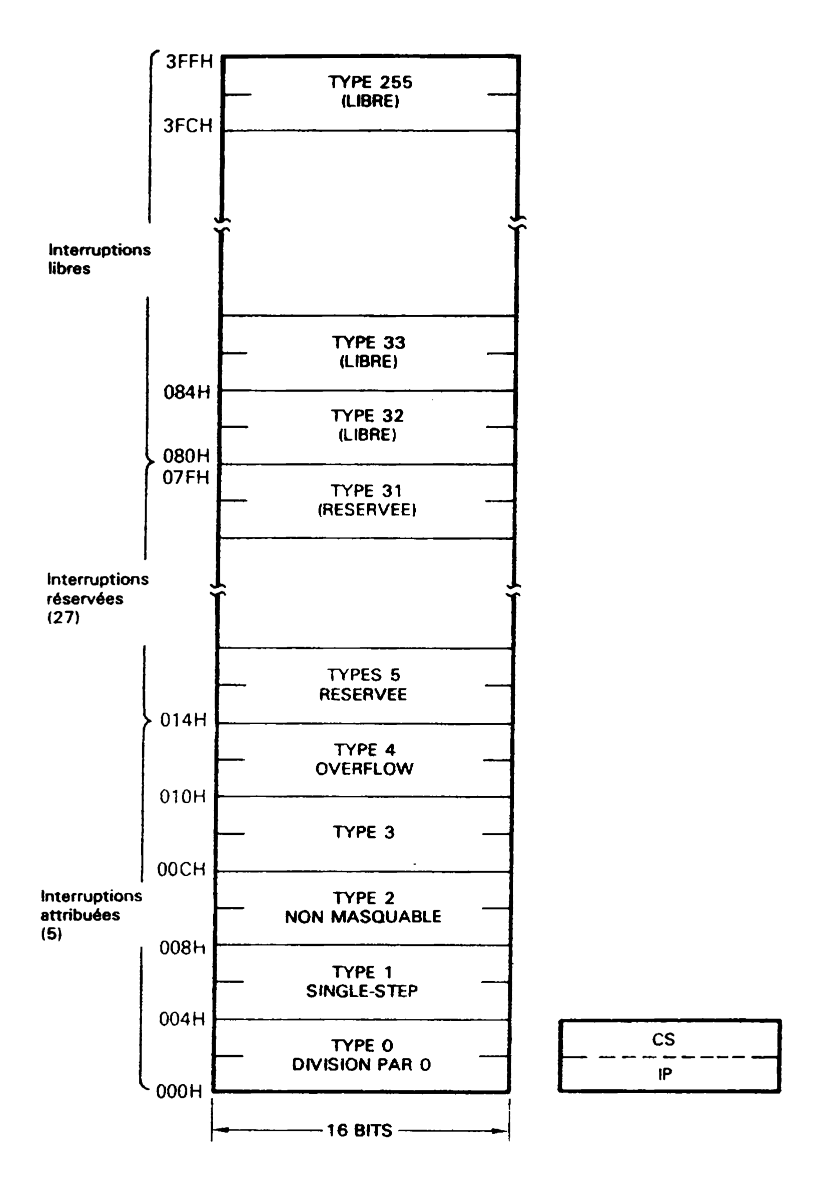


Fig. 6. — Zone mémoire réservée pour les interruptions de 8086/8088.

II.2. — Entrées-sorties, interruptions, zones réservées

Les entrées-sorties, en mode IN/OUT *(I/O MAPPED I/O)* se sont vues réserver les 64 K octets du début de l'espace adressable (les quatre bits de poids fort sont toujours à zéro), les premiers 256 octets peuvent être adressés directement, pour les autres, l'adresse est le contenu de DX.

En cas d'interruptions les flags, les contenus de CS et de IP (dans cet ordre) sont stockés en pile en décrémentant le contenu de SP. L'adresse du début du programme de traitement constituée du contenu de CS et du contenu de IP est rangée (4 octets) dans le premier millier d'octets (1 024) de l'espace adressable (de 0 à 3FFH). Pour 8086 (et 8088), les 20 premiers octets sont attribués comme indiqué au tableau 2.

Interruptions	Adresses	Fonctions
0	00 à 03H	division par 0
1	04 à 08H	pas à pas (TRAP) *Single-Step*
2	084 à 0BH	interruption non masquable (NMI)
3	0CH à 0FH	interruption logicielle sur 1 octet INT 3
4	10H à 13H	interruption sur overflow INTO

Tableau 2. — Interruptions 8086/8088.

Attention, certaines interruptions, internes, sont du genre « *Restart* » du 8085 (appel de sous-programme sur un octet). Ces interruptions sont complétées chez iAPX 186 et iAPX 188 par les suivantes (tableau 3).

Interruptions	Adresses	Fonctions
5	14H à 17H	Dépassement de limites (BOUND)
6	18H à 1BH	Code inconnu
7	1CH à 1FH	Code avec ESCAPE sans extension
8	20H à 23H	Interruption compteur 0
18	48H à 4BH	Interruption compteur 1
19	4CH à 4FH	Interruption compteur 2
9	24H à 27H	Réservée
10	28H à 2BH	Interruption de DMA0
11	2CH à 2FH	Interruption de DMA1
12	30H à 33H	INT0 } Interruptions vectorisées externes masquables
13	34H à 37H	INT1
14	38H à 3BH	INT2
15	3CH à 3FH	INT3
16-31	40H à 7FH	Réservées

Tableau 3. — Interruptions supplémentaires pour iAPX 186 et iAPX 188 (par rapport à 8086).

Note : le code ESCAPE a été créé à l'origine du 8086 pour indiquer des opérations concernant le coprocesseur arithmétique 8087 (voir l'instruction).

Chez iAPX 286, nous retrouvons les interruptions des types 0 à 6 complétées, **en mode réel** par (tableau 4) :

Interruptions	Adresses	Fonctions
5	14H à 17H	Dépassement de limites (BOUND)
6	18H à 1BH	Code inconnu
7	1CH à 1FH	Code avec ESC ou WAIT sans extension
8	20H à 23H	Limite de table d'interruption atteinte (instruction LIDT)
9	24H à 27H	Dépassement de segment pour extension (instruction avec ESC)
13	34H à 37H	Dépassement de segment (AE égale à FFFFH en mode réel)
16	40H à 43H	Erreur de l'extension (instruction WAIT ou ESC)
10-12	28H à 33H	Réservées
14-15	38H à 3FH	Réservées
17-31	44H à 7FH	Réservées

Tableau 4. — Interruptions supplémentaires pour iAPX 286 (par rapport à 8086).

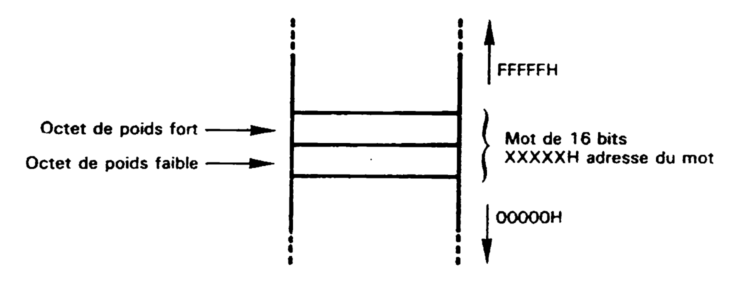
Les interruptions dues à des erreurs de programmation *(exception)* comme le dépassement de segment, se terminent par un retour à l'instruction qui les provoquent. C'est le cas de INT0 (division par 0), INT5 (dépassement des limites), INT6 (code inconnu), INT7 (ESC ou WAIT sans extension) INT8 (interruption hors table) et INT3 (dépassement de segment).
En cas d'interruption externe, le périphérique de contrôle doit fournir au 8086 (8088) le numéro (type) de l'interruption à traiter (voir à ce sujet dans le « *Peripheral Design Handbook* » le chapitre consacré au 8259A).
Une autre zone mémoire est réservée, il s'agit des 16 octets extrêmes de FFFF0H à FFFFFH utilisés par INTEL pour l'adressage de processeur d'entrée/sortie 8089 et le stockage de la première adresse (CS et IP) du programme à traiter après un RESET. En effet, après RESET, les contenus des registres de segment, du pointeur d'instruction et des flags sont (tableau 5) :

	8086/8088	iAPX 186, 188	iAPX 286
Flags	0000H	0002H	0002H
CS	FFFFH	FFFFH	F000H
IP	0000H	0000H	FFF0H
DS	0000H	0000H	0000H
CS	0000H	0000H	0000H
SS	0000H	0000H	0000H

Tableau 5. — Contenu des registres et flags après RESET.
Dans tous les cas, la première adresse est FFFF0H.

II.3. — Mode de travail

Nous n'entrerons pas dans le détail du « matériel » mais, puisque nous travaillons en assembleur, c'est dire que nous désirons écrire des programmes qui « tournent » vite, il faut savoir comment le 8086 lit un mot de 16 bits.
Comme pour 8085, les mots de 16 bits sont rangés avec l'octet de poids fort à l'adresse la plus haute.



Un mot rangé à une adresse *paire* (terminée par 0, 2, 4...) est lu en une fois. Un mot rangé à une adresse *impaire* est lu en deux fois selon le schéma donné par la figure 6. Nous verrons, d'ailleurs, que les temps de lecture diffèrent de 4 périodes d'horloge. Pour lire un mot de 16 bits, **8088** qui a un **bus données de 8 bits** opère en deux temps.

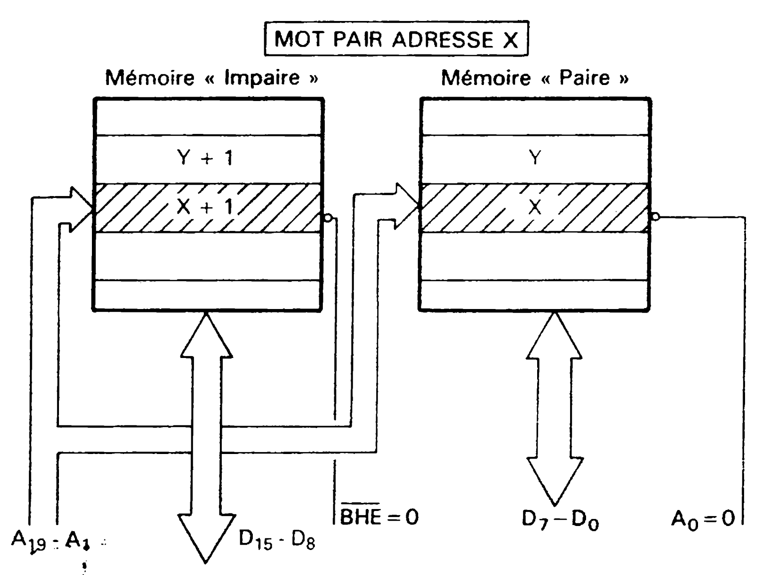
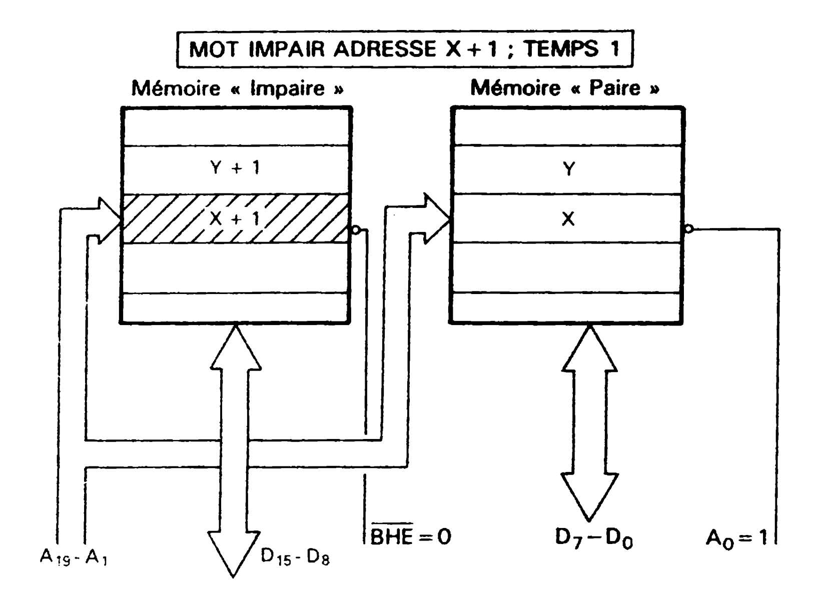


Fig. 6. — Lecture/écriture d'un mot (16 bits) selon son adresse.
(Voir suite page suivante.)



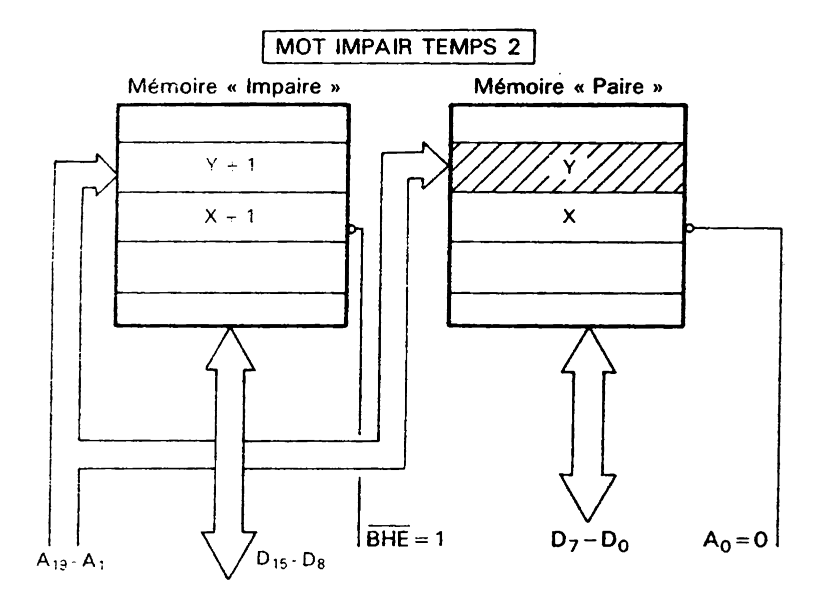


Fig. 6. — Suite et fin

III. — iAPX 186, iAPX 188 et iAPX 286

La figure 7 donne le schéma interne de iAPX 186 et la figure 8 celui de iAPX 286.

On constate que les iAPX 186, iAPX 188 sont constitués d'un 8086 auquel ont été associés :
— 2 canaux de DMA *(Direct Memory Acces* soit Accès Direct Mémoire) ;
— 3 compteurs *(Timer)* 16 bits ;
— 1 contrôleur d'interruptions programmable ;
— 1 unité programmable assurant la sélection des mémoires et des périphériques en associant à ces circuits le nombre de cycles d'attente *(Wait State)* nécessaires.

Quant à iAPX 286, il est surtout orienté vers le traitement multitâches avec protection des accès mémoire.

Ils se présentent en boîtier, carré de 68 contacts.

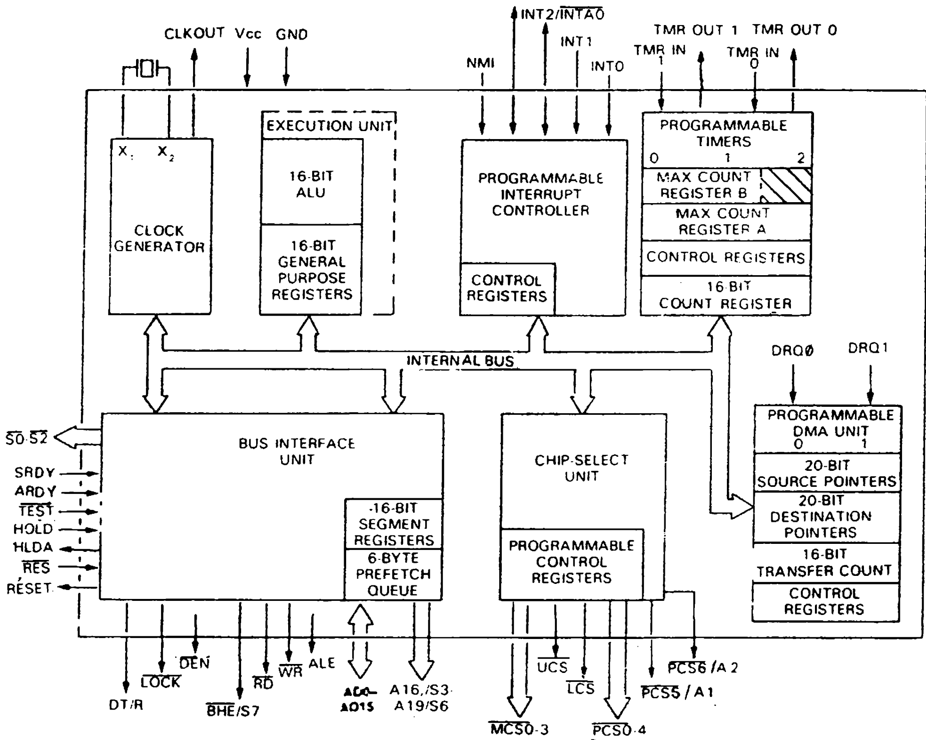


Fig. 7. — **Schéma interne du iAPX 186**

Note : — la queue du iAPX 188 est de 4 octets
— le bus des donnée est de 8 bits

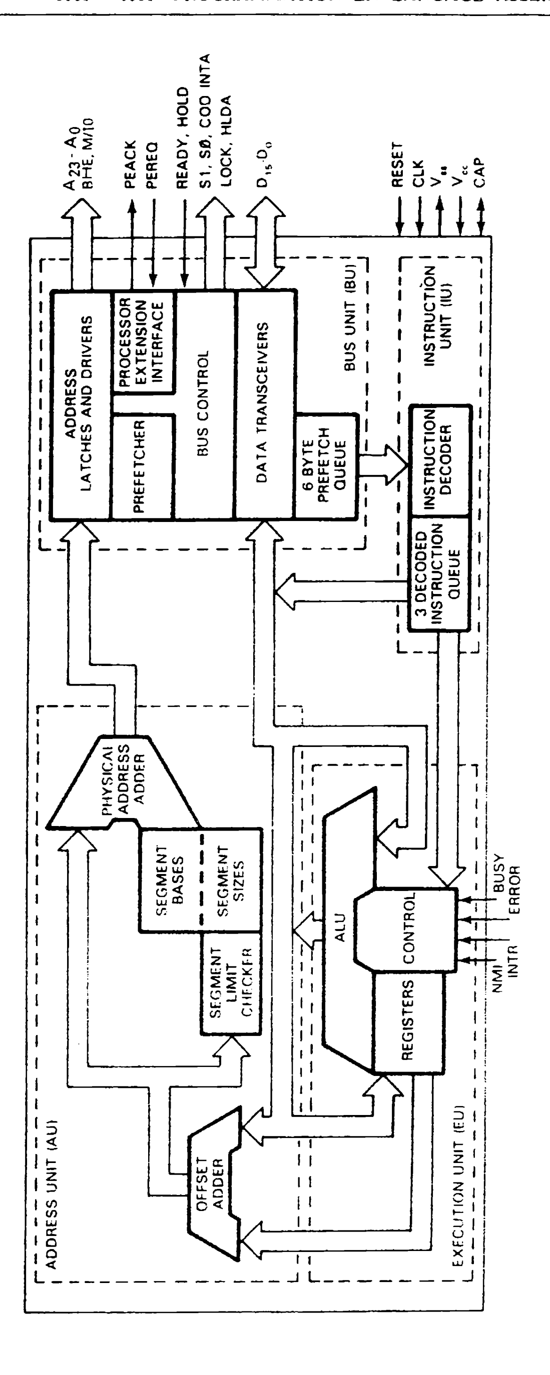


Fig. 8. — Schéma interne du iAPX 286.

INSTRUCTIONS

Chaque instruction est donnée avec son mnémonique, son (ses) code(s) machine, sa (ses) durée(s) *d'exécution (après chargement en file d'attente)* pour les divers processeurs. Pour 8088, les durées sont celles concernant les mots de 16 bits, en 8 bits il travaille à la même vitesse que 8086 sauf pour les opérations ne traitant que des mots de 16 bits (PUSH, POP, JMP, CALL...).
Nous n'avons fait figurer, après une virgule, les durées pour iAPX 188, traitant des mots de 8 bits, que lorsqu'elles diffèrent de celles de iAPX 186. Dans le cas où les mots seraient de 16 bits, il faudrait ajouter 4 périodes par transfert mémoire (écriture ou lecture). Le processeur iAPX 286 est étudié en mode réel.

Nous disposons actuellement de : 8086 travaillant à 5,8 ou 10 MHz (période de 200, 125 et 100 ns) ; 8088 à 5 et 8 MHz ; 186 à 8,10 et 12,5 MHz ; 188 à 8 et 10 MHz ; 286 à 6,8 et 10 MHz ; 386 à 12,5 et 16 MHz (période de 80 et 62,5 ns).

Certaines instructions sont explicitées à l'aide d'applications. Elles sont classées par ordre alphabétique. **Les instructions propres aux iAPX 186 et 186 sont reconnaissables à l'absence de durée pour les autres processeurs.**
En fin de chapitre, on trouvera les instructions spéciales de l'iAPX 286 en mode protégé et un tableau récapitulatif.

I. — Code machine

Une instruction peut compter de 1 à 7 octets. Le code opération peut nécessiter 2 octets. les octets supplémentaires concernent :
— le ou les préfixes : changement de segment, répétition...
— le déplacement, DEP, (8 ou 16 bits), constituant de l'adresse effective AE ;
— la donnée 8 ou 16 bits dans le cas d'opérations *immédiates ;*
— le contenu de CS, de IP, dans le cas d'appel et saut avec changement de segment.

Le deuxième octet du code opération indique :
— le mode d'adressage,
— le ou les registres concernés,
— le type d'instruction dans une famille ayant le même premier octet
selon le schéma suivant :

	mod reg r/m		

où *reg* est un registre, un segment ou le type d'instruction (n), et *mod* associé à *r/m* (*regis*tre/*m*émoire) donne le mode d'adressage (cet octet est souvent appelé ModRM).

I.1. — Codes des registres

Les registres sont codés selon le tableau 6.

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX 001 CX 010 DX 011 BX 100 SP 101 BP 110 SI 111 DI	000 AL 001 CL 010 DL 011 BL 100 AH 101 CH 110 DH 111 BH	00 ES 01 CS 10 SS 11 DS

Tableau 6. — Code des registres.

La distinction entre 16 et 8 bits est faite dans le premier octet à l'aide d'un bit appelé w (w = 1 le mot, le registre traité est de 16 bits).

I.2. — Codes des modes d'adressage

mod : défini l'existence ou non de DEP ou si *r/m* est un registre (reg) ;

r/m : défini le mode de calcul de AE ou est un registre.
Les différentes possibilités sont données par le tableau 7.

— mod = 00	
r/m	AE =
000	(BX) + (SI)
001	(BX) + (DI)
010	(BP) + (SI)
011	(BP) + (DI)
100	(SI)
101	(DI)
110	DEP (16 bits)
111	(BX)

Tableau 7 (début).

— mod=01 ou 10			
r/m	AE=	mod	
		01	10
000	(BX)+SI)+		
001	(BX)+(DI)+	DEP	DEP
010	(BP)+(SI)+		
011	(BP)+(DI)+	8 bits	16 bits
100	(SI)+		
101	(DI)+		
110	(BP)+		
111	(BX)+		
Les 8 bits sont étendus à 16 bits *signés* lors du calcul de AE.			
— mod=11 : r/m=reg			

Tableau 7 (suite).

Les différentes valeurs hexadécimales du Mod RM sont données par le tableau 9. Les durées de calcul de AE sont données au tableau 8.

Composants de AE (Adresse Effective)	
Préfixe de changement de segment	2
Déplacement (DEP)	6
Base ou index	5
Déplacement + base ou index	9
Base + index BP + DI, BX + SI BP + SI, BX + DI	 7 8
Déplacement + base + index BP + DI + DEP, BX + SI + DEP BP + SI + DEP, BX + DI + DEP	 11 12

Tableau 8. — Durées du calcul de l'adresse effective.

R8 =	AL	CL	DL	BL	AH	CH	DH	BH	
R16 =	AX	CX	DX	BX	SP	BP	SI	DI	
n =	0	1	2	3	4	5	6	7	
SEG =	ES	CS	SS	DS					
Valeurs hexadécimales de Mod RM									Adresse effective
	00	08	10	18	20	28	30	38	[BX + SI]
	01	09	11	19	21	29	31	39	[BX + DI]
	02	0A	12	1A	22	2A	32	3A	[BP + SI](2)
	03	0B	13	1B	23	2B	33	3B	[BP + DI](2)
	04	0C	14	1C	24	2C	34	3C	[SI]
	05	0D	15	1D	25	2D	35	3D	[DI]
	06	0E	16	1E	26	2E	36	3E	D16(1)
	07	0F	17	1F	27	2F	37	3F	[BX]
	40	48	50	58	60	68	70	78	[BX + SI] + D8(1)
	41	49	51	59	61	69	71	79	[BX + DI] + D8
	42	4A	52	5A	62	6A	72	7A	[BP + SI] + D8
	43	4B	53	5B	63	6B	73	7B	[BP + DI] + D8
	44	4C	54	5C	64	6C	74	7C	[SI] + D8
	45	4D	55	5D	65	6D	75	7D	[DI] + D8
	46	4E	56	5E	66	6E	76	7E	[BP] + D8(2)
	47	4F	57	5F	67	6F	77	7F	[BX] + D8
	80	88	90	98	A0	A8	B0	B8	[BX + SI] + D16(1)
	81	89	91	99	A1	A9	B1	B9	[BX + DI] + D16
	82	8A	92	9A	A2	AA	B2	BA	[BP + SI] + D16
	83	8B	93	9B	A3	AB	B3	BB	[BP + DI] + D16
	84	8C	94	9C	A4	AC	B4	BC	[SI] + D16
	85	8D	95	9D	A5	AD	B5	BD	[DI] + D16
	86	8E	96	9E	A6	AE	B6	BE	[BP] + D16(2)
	87	8F	97	9F	A7	AF	B7	BF	[BX] + D16
	C0	C8	D0	D8	E0	E8	F0	F8	R16 = AX R8 = AL
	C1	C9	D1	D9	E1	E9	F1	F9	R16 = CX R8 = CL
	C2	CA	D2	DA	E2	EA	F2	FA	R16 = DX R8 = DL
	C3	CB	D3	DB	E3	EB	F3	FB	R16 = BX R8 = BL
	C4	CC	D4	DC	E4	EC	F4	FC	R16 = SP R8 = AH
	C5	CD	D5	DD	E5	ED	F5	FD	R16 = BP R8 = CH
	C6	CE	D6	DE	E6	EE	F6	FE	R16 = SI R8 = DH
	C7	CF	D7	DF	E7	EF	F7	FF	R16 = DI R8 = BH

Tableau 9. — Code hexadécimal de Mod RM.

Note : 1) D16 donnée de 16 bits, D8 donnée de 8 bits qui sera étendue, signée, à 16 bits pour le calcul de AE.

2) Sans préfixe, le registre de segment est SS si le registre BP est utilisé, et DS pour les autres modes d'adressage.

Les durées, variables, selon le mode d'adressage, posent, évidemment, un problème au programmeur qui recherche la vitesse. Ce handicap du 8086/8088 a été levé pour les iAPX 186, 188 et 286 où le temps de calcul de l'adresse effective est fixe, à **une** période près selon qu'elle utilise 2 ou 3 termes.

I.3. — Code du préfixe de changement de segment

Ce code est :

001 reg 110			

en langage machine, cet octet **précède** le premier octet de l'instruction.

I.4. — Code opération

Dans les codes opérations, on rencontre souvent deux bits d et w.

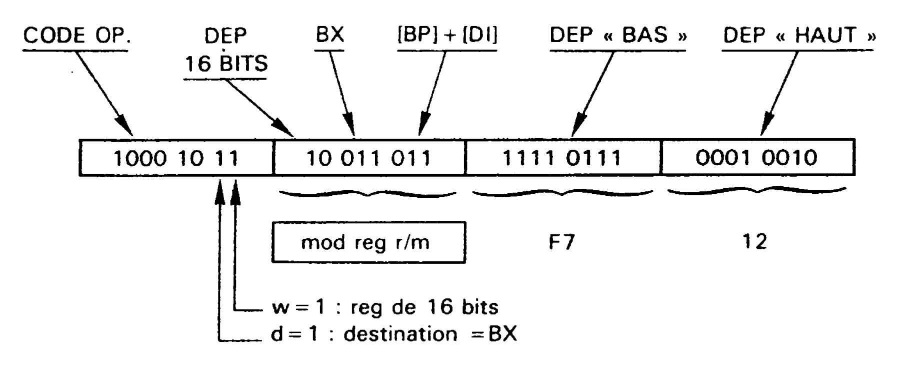
— w : indique la taille des mots traités
 w = 0 ce sont des octets
 w = 1 il s'agit de mots de 16 bits

— d : indique la destination du résultat car il ne s'agit pas toujours de l'accumulateur comme pour 8085.

si d = 1 la destination est un registre, s'il y a deux registres en cause, il s'agit du premier nommé, dans le mnémonique, et il est codé, dans tous les cas par « *reg* ».

I.5. — Exemples de codage

MOV BX, [12 F7] [BP] [DI] *



soit 8B 9B F7 12

* Syntaxe propre à l'assembleur, ligne à ligne, mis au point par l'auteur.

pour MOV [12F7] [BP] [DI], BX
nous aurions :
8A 9B F7 12 (d=0)

La première met le contenu de la case mémoire d'adresse (BP) + (DI) + 12F7 par rapport à (SS) dans BX, la deuxième assure l'opération inverse.
— Si je veux référencer l'adresse par rapport à CS j'écrirai :

MOV BX, CS : [12F7] [BP] [DI]

ce qui donne :

2E 8B 9B F7 12
└ préfixe

— Si je charge directement la case mémoire ci-dessus avec 1284H, j'aurai :

MOV CS : [12F7] [BP] [DI], 1284H

soit

2E C7 83 F7 12 84 12

II. — Présentation des instructions

Chaque mnémonique d'instruction est suivi d'une courte explication sur l'opération qu'elle entraîne.
Un tableau indique :
— **le code machine**



Seuls les octets utiles sont figurés.

Ne pas oublier qu'il comptera un (deux) octet(s) supplémentaire(s) en cas d'adressage mémoire avec DEP, placé(s) **avant** la donnée dans le cas d'opération *immédiate*.

— **la durée** exprimée en nombre de périodes d'horloge. Nous n'avons pas fait figurer « iAPX » dans ces tableaux.

— **les flags** affectés ou indéfinis.
Suivent quelques exemples d'utilisation qui, dans les cas délicats, sont très détaillés.

AAA — Ajustement ASCII pour l'addition

Permet l'addition en chiffres décimaux, écrits en ASCII (codés de 30 à 39) à condition que le contenu de AH soit nul. En effet, cette instruction met systématiquement les 4 bits de poids fort du contenu de AL à 0, et ajoute 6 aux bits de poids faible si le chiffre représenté est supérieur à 9, ou si le flag AF vaut 1, dans ce cas, les flags CF et AF sont mis à 1 et le contenu de AH est incrémenté.

Code machine :

0011 0111			

Durée

8086	8088	186, 188	286
8	8	8	3

Flags : Affectés AF, CF
Indéfinis OF, PF, SF, ZF

Exemples :
si AX contient 2834H, la séquence

```
ADD AL, 37H        codée 04 37
AAA                codée 37
```

conduit à avoir 2901 dans AX en effet

```
     34H  : 0011 0100
   + 37H  : 0011 0111
     6B   : 0110 1011
   + 06   : 0000 0110        à cause de B
     71   : 0111 0001
ET   0F   : 0000 1111
    (AL)  = 0000 0001
```

et (AH) = (AH) + 1 = 29

ce qui revient à avoir : 04 + 07 = 0101 (BCD *unpacked)*.
Il faut donc, au préalable, annuler le contenu de AH.

Remarque : pour cette opération, il n'est pas nécessaire d'annuler, au préalable, le quartet de poids fort du code ASCII.

AAD — Ajustement ASCII pour la division

Prépare le contenu de AX à une division décimale. En effet, elle multiplie par 10D le contenu de AH et l'ajoute au contenu de AL puis met le contenu de AH à 0. On passe nombres BCD *déballé* à des nombres BCD compacts *(packed)*.

Code machine

1101 0101	0000 1010		

Durée

8086	8088	186, 188	286
60	60	15	14

Flags : Affectés PF, SF, ZF
Indéfinis AF, CF, OF

Exemples :
Si le contenu de AX est 0705H après exécution de l'instruction :

AAD codée D5 0A

le contenu de AX sera 004BH (4BH = 75D).

Remarque :
— avant d'être 0705H, le contenu de AX pouvait être 3735H suite à l'entrée des chiffres 7 et 5 codés en ASCII. En fait, 7 et 5 représentent le nombre 75 en DCB « déballé » *(unpacked)*.

— **il faut** au préalable annuler le quartet de poids fort du code ASCII.

AAM — Ajustement ASCII pour la multiplication

Après une multiplication, traduit le contenu de AL en deux chiffres décimaux. Les dizaines sont mises dans AH et les unités dans AL. Permet la multiplication décimale de nombres de 1 chiffre.

Code machine

1101 0100	0000 1010		

Durée

8086	8088	186, 188	286
83	83	19	16

Flags : Affectés PF, SF, ZF
Indéfinis AF, CF, OF

Exemples :

Supposons que le contenu de AL soit 3FH, résultat de la multiplication de 7 par 9 après exécution de l'instruction

AAM codée D4 0A

le contenu de AX sera 0603H.

Note :

On peut remarquer que AAM traduit un nombre hexadécimal en base 10 « déballé » et que AAD réalise l'opération inverse.

Si l'on a accès aux codes machines, on peut écrire des changements de base non prévus par INTEL.

Exemple :

(AX) = 0702

l'instruction : D5 07 donne (AL) = 33

et on remarque que $7 \times 7 + 2 = 51D$ bien que 7 en base 7 soit : 10

$3 \times 16 + 3 = 51D$

l'instruction : D4 06 donne (AX) = 0803

car $8 \times 6 + 3 = 51$ D.

Ainsi l'association D4 X, D5 X, où X représente la base de numération écrite en hexadécimale (0A pour la base 10, 10 pour la base 16) permet de passer respectivement d'un nombre écrit en hexadécimal dans AL au même nombre écrit dans la base X dans AX, et vice versa.

Limite de validité : les chiffres de AX doivent être inférieurs ou égaux à 9 avant D5 X.

AAS — Ajustement ASCII pour la soustraction

Permet la soustraction de chiffres décimaux écrits en ASCII *déballé.* En effet, cette instruction met systématiquement les 4 bits de poids fort du contenu de AL à 0 et retranche 6 aux 4 bits de poids faible si le chiffre représenté est supérieur à 9 ou si AF vaut 1, met CF et AF à 1 et retranche 1 au contenu de AH.

Code machine :

0011 1111			

Durée

8086	8088	186, 188	286
8	8	7	3

Flags : Affectés AF, CF
Indéfinis OF, CF, PF, ZF

Exemples :
Si AX contient 2834, la séquence

SUB AL, 37	codée 2C 37
AAS	codée 3F

conduit à avoir 2707 dans AX, En effet

	34H	=	0011 0100
	−37H	=	1100 1001
	FDH	=	1111 1101
	−06H	=	1111 1010
	F7H	=	1111 0111
ET	F0H	=	1111 0000
	07H	=	0000 0111

et (AH) = (AH) − 1 = 27
Nous sommes prévenus d'un résultat négatif par la décrémentation de (AH).

Remarques :
— En soustraction, le carry est complémenté à 1 ainsi que la retenue intermédiaire. Dans l'exemple ci-dessus, avant l'exécution de l'instruction AAS, nous avons :

CF = 1 et AF = 1

— Il n'est pas nécessaire pour cette opération d'annuler les quartets de poids fort du code ASCII.

ADC — Addition avec retenue (carry)

Cette instruction réalise l'addition entre les contenus des deux opérandes, ajoute 1 au résultat si le flag CF était à 1 et stocke le résultat dans la case mémoire ou le registre indiqué par le premier opérande.
Ainsi, si le contenu de AL est 4C et celui de la case mémoire d'adresse symbolique BETA est B2 après exécution de l'instruction

ADC BETA, AL

la case mémoire BETA (référencée par rapport à DS) contiendra FE si CF était nul.
Par contre,

ADC AL, BETA

mettra le résultat dans AL.
En code machine, cette instruction possède 3 codages selon la nature des opérandes.

1. Mémoire ou registre avec registre

Code machine

0001 00dw	mod reg r/m		

d = 1 : le résultat doit être mis dans le registre défini par *reg* ; s'il y a deux registres, il s'agit du premier nommé.

Durée

	8086	8088	186, 188	286
registre + registre + CF	3	3	3	2
registre + mémoire + CF	9 + AE	13 + AE	10	7
mémoire + registre + CF	16 + AE	24 + AE	10	7

Flags : Affectés OF, SF, ZF, AF, PF, CF
Indéfinis

Exemples :

ADC CH, BC codée 10 EB
ADC AX, BETA codée 13 selon code BETA
ADC BETA, AX codée 11 selon code BETA

2. Addition immédiate dans l'accumulateur (AX ou AL)

Code machine

0001 010w	donnée	donnée si w = 1	

Durée

8086	8088	186, 188	286
4	4	3-4	3

Flags : Affectés OF, SF, ZF, AF, PF, CF
Indéfinis

Exemples :

ADC AL,3 codée 14 03
ADC AX,3 codée 15 03 00
ADC AH,3 n'existe pas sous cette forme

3. Addition immédiate au contenu d'une case mémoire ou d'un registre

Code machine

1000 00sw	mod 010 r/m	donnée	donnée

si s = 0 et w = 1

Durée

	8086	8088	186, 188	286
registre	4	4	4	3
mémoire	17+AE	25+AE	16	7

Flags : Affectés OF, SF, ZF, AF, PF, CF
Indéfinis

Exemples :

```
ADC BETA,4
ADC CX,432H
```

Note :
si on ajoute un octet à un mot (16 bits), la donnée est étendue à 16 bits signée, et s et w sont mis à 1.

```
ADC AH,03        codée : 80 D4 03
ADC BH,03        codée : 80 D7 03
ADC BX,03        codée : 83 D3 03
ADC BX,0103      codée : 81 D3 03 01
```

ADD — Addition

Les contenus des deux opérandes sont additionnés, le résultat est stocké dans la case mémoire, ou le registre indiqué par le premier opérande.
Comme ADC, en code machine, cette instruction possède 3 codages selon la nature des opérandes.

1. Mémoire ou registre avec registre

Code machine

0000 00dw	mod reg r/m		

d = 1 : le résultat sera mis dans le registre défini par *reg*. S'il y a 2 registres, il s'agit du premier nommé.

Durée

	8086	8088	186, 188	286
registre + registre	3	3	3	2
registre + mémoire	9+AE	13+AE	10	7
mémoire + registre	16+AE	24+AE	10	7

Flags : Affectés OF, SF, ZF, PF, AF, CF
Indéfinis

Exemples :

ADD CH,BL codée 02 EB
ADD AX,BETA codée 03 selon code BETA
ADD BETA,AX codée 01 selon code BETA

2. Addition immédiate dans l'accumulateur (AL ou AX)

Code machine

0000 010w	donnée	donnée si w = 1	

Durée

8086	8088	186, 188	286
4	4	3-4	3

Flags : Affectés OF, SF, ZF, AF, PF, CF
Indéfinis

Exemples :

ADD AL,3 codée 04 03
ADD AX,1FC4H codée 05 C4 1F

3. Addition immédiate au contenu d'une case mémoire ou d'un registre

Code machine

1000 000sw	mod 000 r/m	donnée	donnée

si s = 0 et w = 1

Durée

	8086	8088	186, 188	286
registre	4	4	4	3
mémoire	17 + AE	25 + AE	16	7

Flags : Affectés OF, SF, ZF, AF, PF, CF
Indéfinis

Exemples :

```
ADD BETA,4
ADD CX,432
```

Note :
si on ajoute un octet à un mot (16 bits), la donnée est étendue à 16 bits, signée, et s et w sont mis à 1.

AND — ET (logique)

Réalise un ET entre les contenus des deux opérandes, le résultat est stocké dans la case mémoire ou le registre indiqué par le premier opérande.
En code machine, cette instruction possède 3 codages selon la nature des opérandes.

1. Mémoire ou registre avec registre

Code machine

0010 00dw	mod reg r/m		

Si d = 1, le résultat doit être mis dans le registre défini par « *reg* », s'il y a deux registres, il s'agit du premier nommé.

Durée

	8086	8088	186, 188	286
entre registres	3	3	3	3
registre ET mémoire	9 + AE	13 + AE	10	7
mémoire ET registre	16 + AE	24 + AE	10	7

Flags : Affectés PF, SF, ZF, CF = 0, OF = 0
Indéfinis AF

Exemples :

AND CH,BC codée 22 EB
AND AX,BETA codée 23 selon code de BETA
AND BETA,AX codée 21 selon code de BETA

2. ET immédiat dans l'accumulateur

Code machine

0010 010w	donnée	donnée si w = 1	

Durée

8086	8088	186, 188	286
4	4	3-4	3

Flags : Affectés PF, SF, ZF, CF = 0, OF = 0
Indéfinis AF

Exemples :

AND AL, OFH codée 24 OF
AND AX, OOFFH codée 25 FF 00

3. ET immédiat avec contenu de case mémoire ou de registre

Code machine

1000 000w	mod 100 r/m	donnée	donnée si w = 1

Durée

	8086	8088	186, 188	286
registre	4	4	4	3
mémoire	17+AE	25+AE	16	7

Flags : Affectés PF, SF, ZF, CF=0, OF=0
Indéfinis AF

Exemples :

AND CX, 432 H codée 81 EI 32 04
AND BETA, 0FH

BOUND — Limite

Cette instruction permet de détecter le dépassement de valeurs limites, signées, inférieures et supérieures du contenu d'un registre (il peut y avoir égalité).
Les valeurs limites ont été, préalablement, stockées en mémoire, la borne supérieure suivant **immédiatement** la borne inférieure. Si le contenu du registre est hors des limites fixées, une interruption de type 5 est générée.

Code machine

0110 0010	mod reg r/m		

Durée

8086	8088	186, 188	286
—	—	33/35	13

Flags : Affectés Aucun
Indéfinis

Exemples :

BOUND BX,INF

Si (BX) est plus petit que (INF) ou plus grand que (INF + 2), il y a interruption de type 5.
En général, les valeurs limites sont stockées en début de table de donnée et on écrit BOUND BX, TABLE – 4

CALL — Appel de sous-programme (procédure)

Cette instruction permet l'appel d'un sous-programme que celui-ci dépende du même segment (tranche de 64 K octets) ou non. Lors d'un appel, le « compteur ordinal » *(Instruction Pointer :* IP) est stocké en pile après stockage éventuel du contenu de CS. L'adressage peut être direct, relatif ou indirect.

1. Appel intra segment (CS inchangé) - **Appel relatif**
(SP) est décrémenté de 2

Code machine

1110 1000	dec. bas	dec. haut	

dec. bas :octet de poids faible au décalage
dec. haut : octet de poids fort au décalage

L'appel a lieu pour un programme commençant en (IP) + DEC où (IP) est l'adresse de l'instruction qui **suit** l'appel.

Durée

8086	8088	186, 188	286
19	23	14, 18	7 + m*

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

```
CALL AFF_DONNEES
```

seul (IP) est stocké en pile, (SP) → (SP) − 2
AFF_DONNEES a été définie comme une procédure *proche (NEAR)*

2. Appel inter segment direct (CS changé)

(SP) est décrémenté de 4

Code machine

1001 1010	IP Bas	IP Haut	SEG Bas	SEG Haut

(IP) prend la valeur IP Haut, IP Bas
(CS) prend la valeur SEG Haut, SEG Bas

Durée

8086	8088	186, 188	286
28	36	23, 31	13+m*

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

CALL FAR_PROC

(IP) et (CS) sont stockés en pile, (SP) → (SP) − 4

3. Appel intra segment indirect (CS inchangé)
(SP) est décrémenté de 2

Code machine

1111 1111	mod 010 r/m		

Durée

	8086	8088	186, 188	286
registre	16	24	13, 17	7+m*
mémoire	21+AE	29+AE	19, 27	11+m*

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

```
CALL WORD PTR*[BX]        codée FF 17
CALL BX                   codée FF D3
```

Dans le premier cas, la nouvelle valeur de (IP) sera le contenu de la case mémoire (16 bits = WORD) dont l'**adresse** est le **contenu** de **BX par rapport à DS.** Dans le deuxième cas, la nouvelle valeur de (IP) sera le contenu de BX.

* Voir note page suivante

4. Appel inter segment indirect (CS changé)
(SP) est décrémenté de 4

Code machine

1111 1111	mod 011 r/m		

mod ≠ 11

(IP) et (CS) sont remplacés respectivement par les contenus des cases mémoire (16 bits) définies par l'adresse effective et l'adresse effective + 2.

Durée

8086	8088	186, 188	286
37 + AE	57 + AE	38, 54	16 + m*

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :
Supposant que le contenu de DS soit 0400H et celui de BX 0006H, que le contenu de la case mémoire d'adresse 4006H soit 0100H et celui de la case mémoire d'adresse 4008 soit 0FA0H, après exécution de

CALL DWORD PTR [BX]

le contenu de IP sera 0100H et celui de CS 0FA0H le programme continuera à partir de l'adresse vraie 0FB00H (0FA00 + 100).

Note :
PTR opérateur de ASM86, permet de préciser, ici, que l'on prend un mot (16 bits) ou deux pointés par [BX].
Les appels **inter segment** exigent que le sous-programme appelé se termine par un **retour long.**

CBW — Convertir 8 bits (bytes) en 16 bits (word)

Traduit le contenu de AL en un nombre signé de 16 bits dans AX. C'est-à-dire que AH contiendra 00 si le bit de poids fort de (AL) est 0 et FF dans le cas contraire.

Code machine

1001 1000			

Durée

	8086	8088	186, 188	286
registre	4	4	4	3
mémoire	17+AE	25+AE	16	7

Flags : Affectés PF, SF, ZF, CF=0, OF=0
Indéfinis AF

Exemples :

AND CX, 432 H codée 81 E1 32 04
AND BETA, 0FH

BOUND — Limite

Cette instruction permet de détecter le dépassement de valeurs limites, signées, inférieures et supérieures du contenu d'un registre (il peut y avoir égalité).
Les valeurs limites ont été, préalablement, stockées en mémoire, la borne supérieure suivant **immédiatement** la borne inférieure. Si le contenu du registre est hors des limites fixées, une interruption de type 5 est générée.

Code machine

0110 0010	mod reg r/m		

Durée

8086	8088	186, 188	286
—	—	33/35	13

Flags : Affectés Aucun
Indéfinis

Exemples :

BOUND BX,INF

Si (BX) est plus petit que (INF) ou plus grand que (INF+2), il y a interruption de type 5.
En général, les valeurs limites sont stockées en début de table de donnée et on écrit BOUND BX, TABLE−4

CALL — Appel de sous-programme (procédure)

Cette instruction permet l'appel d'un sous-programme que celui-ci dépende du même segment (tranche de 64 K octets) ou non. Lors d'un appel, le « compteur ordinal » *(Instruction Pointer :* IP) est stocké en pile après stockage éventuel du contenu de CS. L'adressage peut être direct, relatif ou indirect.

1. Appel intra segment (CS inchangé) - **Appel relatif**
(SP) est décrémenté de 2

Code machine

1110 1000	dec. bas	dec. haut	

dec. bas :octet de poids faible au décalage
dec. haut : octet de poids fort au décalage

L'appel a lieu pour un programme commençant en (IP) + DEC où (IP) est l'adresse de l'instruction qui suit l'appel.

Durée

8086	8088	186, 188	286
19	23	14, 18	7+m*

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

```
CALL AFF_DONNEES
```

seul (IP) est stocké en pile, (SP) → (SP) − 2
AFF_DONNEES a été définie comme une procédure *proche (NEAR)*

2. Appel inter segment direct (CS changé)

(SP) est décrémenté de 4

Code machine

1001 1010	IP Bas	IP Haut	SEG Bas	SEG Haut

(IP) prend la valeur IP Haut, IP Bas
(CS) prend la valeur SEG Haut, SEG Bas

Durée

8086	8088	186, 188	286
28	36	23, 31	13+m*

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

CALL FAR_PROC

(IP) et (CS) sont stockés en pile, (SP) → (SP) − 4

3. Appel intra segment indirect (CS inchangé)
(SP) est décrémenté de 2

Code machine

1111 1111	mod 010 r/m		

Durée

	8086	8088	186, 188	286
registre	16	24	13, 17	7+m*
mémoire	21+AE	29+AE	19, 27	11+m*

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

CALL WORD PTR*[BX] codée FF 17
CALL BX codée FF D3

Dans le premier cas, la nouvelle valeur de (IP) sera le contenu de la case mémoire (16 bits = WORD) dont l'**adresse** est le **contenu de BX par rapport à DS.**
Dans le deuxième cas, la nouvelle valeur de (IP) sera le contenu de BX.

* Voir note page suivante

4. Appel inter segment indirect (CS changé)
(SP) est décrémenté de 4

Code machine

1111 1111	mod 011 r/m		

mod ≠ 11

(IP) et (CS) sont remplacés respectivement par les contenus des cases mémoire (16 bits) définies par l'adresse effective et l'adresse effective +2.

Durée

8086	8088	186, 188	286
37+AE	57+AE	38, 54	16+m*

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :
Supposant que le contenu de DS soit 0400H et celui de BX 0006H, que le contenu de la case mémoire d'adresse 4006H soit 0100H et celui de la case mémoire d'adresse 4008 soit 0FA0H, après exécution de

CALL DWORD PTR [BX]

le contenu de IP sera 0100H et celui de CS 0FA0H le programme continuera à partir de l'adresse vraie 0FB00H (0FA00+100).

Note :
PTR opérateur de ASM86, permet de préciser, ici, que l'on prend un mot (16 bits) ou deux pointés par [BX].
Les appels **inter segment** exigent que le sous-programme appelé se termine par un **retour long.**

CBW — Convertir 8 bits (bytes) en 16 bits (word)

Traduit le contenu de AL en un nombre signé de 16 bits dans AX. C'est-à-dire que AH contiendra 00 si le bit de poids fort de (AL) est 0 et FF dans le cas contraire.

Code machine

1001 1000			

Durée

8086	8088	186, 188	286
2	2	2	2

Flags : Affectés aucun
Indéfinis

Exemples :
— si AL contient 68H après exécution de CBW, nous aurons : (AX) = 0068H
— si AL contient 83H après exécution de CBW, nous aurons : (AX) = FF83H

CLC — Mettre le carry à zéro (clear)

Force le flag CF à 0.

Code machine

1111 1000			

Durée

8086	8088	186, 188	286
2	2	2	2

Flags : Affectés CF
Indéfinis

Exemples :

CLC codée FE

CLD — Mettre le flag direction à zéro

Force le flag DF à zéro, ce qui permet un travail sur suite de caractères avec *auto-incrémentation* des adresses.

Code machine

1111 1100			

Durée

8086	8088	186, 188	286
2	2	2	2

Flags : Affectés DF
Indéfinis

Exemples :

CLD codée FC

CLI — Mettre le flag d'interruption à zéro

Met le flag IF à zéro interdisant les interruptions externes, sauf l'interruption non masquable (NMI).

Code machine

1111 1010			

Durée

8086	8088	186, 188	286
2	2	2	2

Flags : Affectés IF
Indéfinis

Exemples :

CLI codée FA

CMC — Complémenter le flag CF

Complémente le flag CF qui sera mis à 1 s'il valait 0 et vice versa.

Code machine

1111 0101			

Durée

8086	8088	186, 188	286
2	2	2	2

Flags : Affectés CF
Indéfinis

Exemples :

CMC codée F5

CMP — Comparaison des contenus de deux opérandes

Compare les contenus de deux opérandes en effectuant une soustraction (le second soustrait du premier). Le résultat n'est pas retourné, seuls les flags reflètent la comparaison.

Contrairement au 8085 où l'arithmétique n'est pas signée et où seuls les flags Z et C sont significatifs, nous devons dans le cas de nombres signés examiner les flags OF et SF. Les flags sont affectés selon le tableau suivant :
N_1 est comparé à N_2, c'est-à-dire que l'on effectue $(N_1 - N_2)$.

		OF⊕SF	ZF	CF
N_1 et N_2 positifs	$N_1 > N_2$	0	0	0
	$N_1 = N_2$	0	1	0
	$N_1 < N_2$	1	0	1
N_1 et N_2 négatifs	$N_1 > N_2$	0	0	0
	$N_1 = N_2$	0	1	0
	$N_1 < N_2$	1	0	1
N_1 négatif, N_2 positif :	$N_1 < N_2$	1	0	0
N_1 positif, N_2 négatif :	$N_1 > N_2$	0	0	1

Nous trouverons donc les expressions suivantes : inférieur (supérieur) à... pour l'arithmétique non signée et plus petit (plus grand) que... par l'arithmétique signée, c'est-à-dire :

inférieur à : CF = 1
plus petit que : 0F ⊕ SF = 1
supérieur ou égal à : CF = 0
plus grand que : 0F ⊕ SF = 0

En code machine, nous avons trois possibilités selon la nature des opérandes.

1. Mémoire ou registre, avec registre

Code machine

0011 10 dw	mod reg r/m		

d = 1. Le premier opérande est celui défini par *reg*. S'il y a deux registres, il s'agit du premier nommé.

Durée

	8086	8088	186, 188	286
entre registres	3	3	3	2
entre registres et mémoire	9+AE	13+AE	10	6

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

CMP AX,DX codée 3B C2

CMP BETA,CX codée 39 selon code BETA

CMP CX,BETA codée 3B selon code BETA

2. Immédiate avec l'accumulateur (AL ou AX)

Code machine

0011 110w	donnée	donnée si w = 1	

Durée

8086	8088	186, 188	286
4	4	3-4	3

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

CMP AL,03H codée 3C 03
CMP AX,03H codée 3D 03 00

3. Immédiate avec un registre ou une case mémoire

Code machine

1000 00sw	mod 111 r/m	donnée	donnée si s=0 et w=1

Durée

	8086	8088	186, 188	286
registre	4	4	3	3
mémoire	10+AE	14+AE	10	6

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

CMP BX,798H codée 81 FB 98 07

CMP BETA,6ACEH codée 81 selon code BETA

CMP AH,03H codée 80 FC 03

Note : si une donnée de 8 bits doit être comparée à une donnée de 16 bits, elle sera étendue à 16 bits, signée, et s mis à 1 ainsi que w.

CMP BX,12H codée 83 FB 12

CMPS — Comparaison de suites de mots

Permet la comparaison de deux données, l'une d'adresse définie par (SI), par rapport à (DS), l'autre qui lui est soustraite, d'adresse définie par (DI) par rapport à (ES).

Après la comparaison (SI) et (DI) sont incrémentés, si le flag DF est égal à 0 (l'instruction CLD a été exécutée), de 1 si les données sont des octets, de 2 si elles sont de 16 bits. Cette instruction peut être répétée si elle est précédée du *préfixe* REP (REPNZ ou REPZ).

Code machine

1010 011w			

w=0, les données sont de 8 bits

Durée

	8086	8088	186, 188	286
une opération n opérations	22 9+22*n	30 9+30*n	22 5+22*n	8 5+9*n

Flags : Affectés OF, SF, ZF, AF, PF, CF
Indéfinis

Exemples :

CMPS SERIE1, SERIE2 codée A6 ou A7

REPE CMPS CODE, ENTREE codée F3 A6 ou A7

La taille des mots traités (8 ou 16 bits) a été préalablement définie.

CWD — Convertir un mot de 16 bits (word) en un mot de 32 bits (double word)

Convertit le contenu de AX en un mot de 32 bits contenu dans DX pour le poids fort et AX en mettant FFFF dans DX si le contenu de l'accumulateur est négatif (bit de poids fort à 1) sinon, le contenu de DX sera nul.

Cette instruction précéde généralement une division.

Code machine

1001 1001			

Durée

8086	8088	186, 188	286
5	5	4	2

Flags : Affectés aucun
Indéfinis

Exemples :

CWD codée 99

DAA — Ajustement décimal pour l'addition

Cette instruction identique à celle du 8085, convertit le contenu de AL, résultat d'une addition de nombres décimaux en un nombre décimal, en ajoutant 6 à un quartet s'il est plus grand que 9, ou si la retenue correspondante (AF pour le quartet de poids faible, CF pour le quartet de poids fort) vaut 1.

Code machine

0010 0111			

Durée

8086	8088	186, 188	286
4	4	4	3

Flags : Affectés AF, CF, PF, SF, ZF
Indéfinis OF

Exemples :
Supposons que le contenu de AL soit 5E résultat de l'addition de 29 à 35 après exécution de

DAA codée 27

ce contenu sera 64.

```
          29     0010 1001
         +35    +0011 0101
          5E     0101 1110

E>9      +06     0000 0110
          64     0110 0100
```

Note : à utiliser immédiatement après une addition (ADD ou ADC) dans AL de nombres décimaux si l'on veut un résultat décimal.

DAS — Ajustement décimal pour la soustraction

Convertit le contenu de l'accumulateur, AL, résultat d'une soustraction entre nombres décimaux en un nombre décimal, en retirant 6 à un quartet s'il est plus grand que 9 ou si la retenue correspondante est à 1.

Code machine

0010 1111			

Durée

8086	8088	186, 188	286
4	4	4	3

Flags : Affectés SF, ZF, AF, PF, CF
Indéfinis OF

Exemples :

1. Supposons que AL contienne 3E résultat de la soustraction de 27 à 65 après exécution de

DAS codée 2F

le contenu de AL sera 28, en effet

			1 0	
	65		0011 0101	
complément à 2 de	27	:	1101 1001	
	3E	:	0 0011 1110	en soustraction CF→$\overline{CF}$ et AF→$\overline{AF}$
complément à 2 de	+06	:	1111 1010	
	38	:	0011 1000	

2. Supposons que AL contienne D3 résultat de la soustraction de 92 à 65, après exécution de DAS, le contenu de AL sera 73 !

		1 1	
	65	0110 0101	
complément à 2 de	92	0110 1110	
	D3	1 1101 0011	en soustraction CF→$\overline{CF}$ et AF→$\overline{AF}$
complément à 2 de	60	1010 0000	comme AF = 0 on retire 60
	73	0111 0011	

Soit le complément à 100 de 27 ! Nous sommes prévenus de cette perte d'information par un OF à 1.

DEC — Retirer 1 (décrémenter)

Retire 1 au contenu d'un registre ou d'une case mémoire.
En code machine, il y a 2 possibilités.

1. Décrémenter le contenu (16 bits) d'un registre

Code machine

0100 1 reg			

Durée

8086	8088	186, 188	286
3	3	3	2

Flags : Affectés AF, OF, PF, SF, ZF
Indéfinis

Exemples :

DEC AX codée 48

DEC SI codée 4E

2. Décrémenter le contenu (8 ou 16 bits) d'un registre ou d'une case mémoire

Code machine

1111 111w	mod 001 r/m		

si w = 1, la donnée décrémentée est de 16 bits

Durée

	8086	8088	186, 188	286
registre	3	3	3	3
mémoire	15+AE	23+AE	15	7

Flags : Affectés AF, OF, PF, SF, ZF
Indéfinis

Exemples :

DEC AL	codée FE C8
DEC MEM_BYTE	codée FE selon code MEM_BYTE
DEC MEM_WORD	codée FF selon code MEM_WORD

Note : DEC n'affecte pas CF.

DIV — Division non signée

Réalise la division **non signée** d'un nombre de 16 bits, contenu dans AX (ou de 32 bits, contenu dans DX et AX) par le nombre de 8 ou 16 bits, indiqué par le deuxième opérande. Le quotient est stocké dans AL (ou AX) et le reste dans AH (ou DX).

La valeur maximale du quotient étant FF (ou FFFF), si cette valeur est dépassée, il y a génération d'une interruption de type 0. Les flags, (CS) et (IP) sont stockés en pile, (CS) prend la valeur rangée en 2H (16 bits) et IP celle rangée en 0H (16 bits) ; (SP) est décrémenté de 6 ; les flags IF et TF sont mis à 0.

Le second opérande (diviseur) peut être le contenu d'un registre ou d'une case mémoire.

Code machine

1111 011w	mod 110 r/m		

w = 0 DIVIDENDE = (AX)
QUOTIENT = (AL) FFH
RESTE = (AH)

w = 1 DIVIDENDE = (DX,AX)
QUOTIENT = (AX) FFFFH
RESTE = (DX)

Durée

	8086	8088	186, 188	286
- 8 bits : registre	80 – 90	80 – 90	29	14
mémoire	(86-96) + AE	(86-96) + AE	35	17
- 16 bits : registre	144-162	144-162	38	22
mémoire	(154 – 172) + AE	(158 – 176) + AE	44, 48	25

Flags : Affectés tous mais non significatifs
Indéfinis

Exemples :

— 1. 8 bits

DIV BL	codée F6 F3
DIV BYTE_BETA	codée F6 selon code BYTE_BETA

— 2. 16 bits

DIV BX	codée F7 F3
DIV WORD_BETA	codée F7 selon code WORD_BETA

Note : avant une division, il faut utiliser soit CBW (8 en 16 bits) soit CWD (16 en 32 bits).

ENTER — Entrer dans une procédure

Cette instruction en début de programme permet de réserver un certain nombre d'octets en pile pour le stockage dynamique des données et de définir le niveau d'emboîtement *(lexical nesting level)* dans le cas d'enchaînement de procédures.

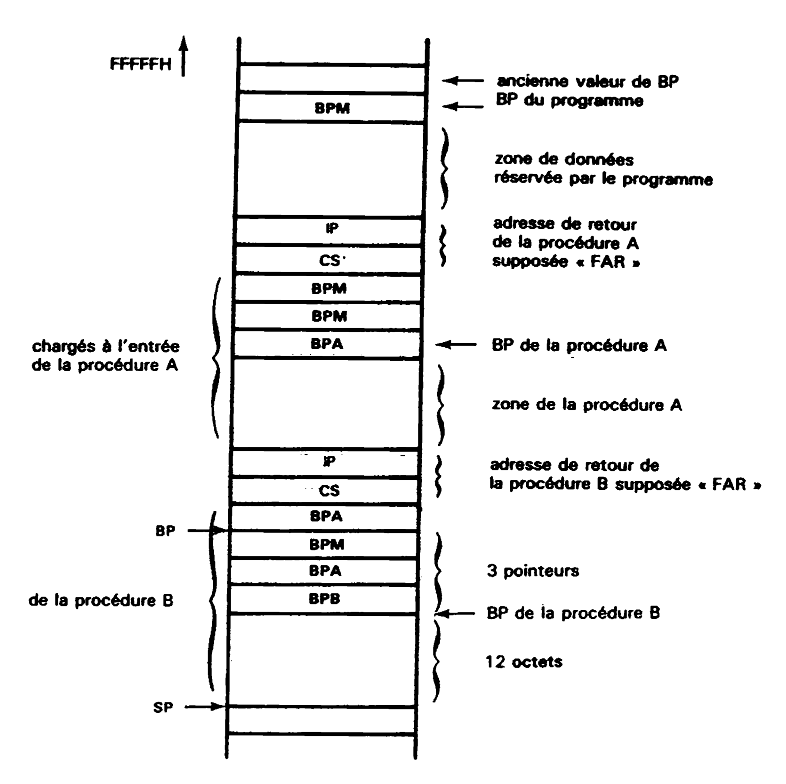
Dans une procédure commençant par ENTER, le registre **BP sert de pointeur de donnée.** Le niveau d'emboîtement détermine le nombre de pointeurs (valeurs du contenu de BP) qui sont stockés en haut de la zone mémoire réservée (adresse la plus haute). Il s'agit des pointeurs des procédures ou programmes appelant la procédure en cours.

Les opérations de sauvegarde et de restauration étant toujours possibles sont gérées par SP, le contenu de ce registre est ajusté de façon à pointer le dernière case mémoire (16 bits) réservée.

Par exemple, la procédure B commence par

ENTER 12,3

Elle a été appelée par la procédure A de niveau 2, elle-même appelée par un programme de niveau 1. Après exécution de l'instruction ENTER 12,3 nous avons en mémoire les données suivantes :



L'ensemble des pointeurs stockés est quelques fois appelé *display*. Pour la procédure B, ils sont 4 (BPA, BPM, BPA, BPB).

Le déroulement des opérations est le suivant :

```
PUSH BP
TAMPON = SP
si NIVEAU (Level) > 0
    BP = BP - 2   }
    PUSH [BP]     }  répéter (Niveau - 1) fois
    PUSH TAMPON
BP = TAMPON
SP = SP - 1er opérande
```

Le niveau doit être inférieur à 32 sinon seuls les 5 bits de plus faible poids sont pris en compte.

Code machine

1100 1000	DONNEE BASSE	DONNEE HAUTE	N = NIVEAU

Durée

	8086	8088	186, 188	286
NIVEAU = 0	—	—	15	11
NIVEAU = 1	—	—	25	15
NIVEAU > 1	—	—	22 + 16(N − 1)	16 + 4(N − 1)

Flags : Affectés aucun
Indéfinis

Exemple :

ENTER 2048,3 codée C8 00 08 03

ESC — Escape

Cette instruction permet d'inclure dans un programme des ordres ne concernant pas le 8086 mais destinés à un autre processeur (coprocesseur) comme le 8087 ou NPX *(Numeric Processor Extension).*
Si l'opérande est un registre de 16 bits (mod = 11), l'instruction est équivalente à NOP pour le 8086. L'opération se réduit au chargement du bus à l'aide d'un ordre ; par contre, si l'opérande est une case mémoire, le processeur *maître* continue son travail, c'est-à-dire qu'il va lire le contenu de la case mémoire concernée, mettant sa valeur sur le bus à destination du coprocesseur.

Code machine

1101 1 $b_5b_4b_3$	mod $b_2b_1b_0$ r/m		

$b_5b_4b_3b_2b_1b_0$ = 1er opérande

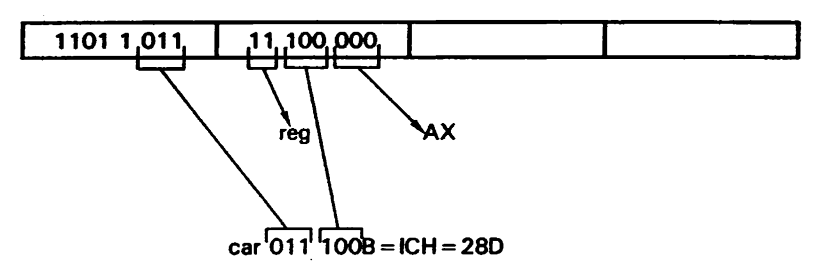
Durée

	8086	8088	186, 188	286
registre	2	2	2	9
mémoire	8 + AE	12 + AE	6	20

Flags : Affectés aucun
Indéfinis

Exemples :
concernant le 8087

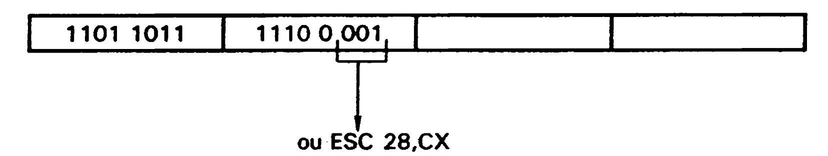
— ESC 28,AX codée DB E0



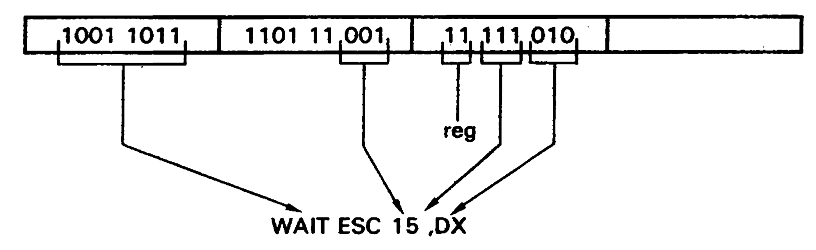
équivaut pour 8087 à : FNENI, c'est-à-dire : mise à zéro du masque d'interruption du mot de contrôle permettant au 8087 de générer une demande d'interruption. Cette instruction n'est pas précédée de WAIT, son homologue avec WAIT est FENI soit :

WAIT ESC 28,AX codée 9B DB E0

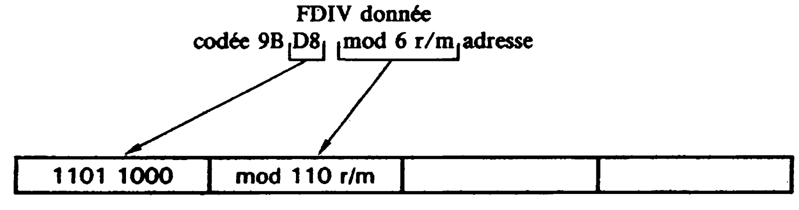
— Pour mettre le masque, on utilise FDISI ou FNDISI codée DB EI soit :



— L'extraction de la racine carrée est FSQRT codée 9B D9 FA soit :



— La division du mot du haut de la pile de 8087 par une donnée (en « *réel court* », écriture scientifique sur 32 bits) est :



soit WAIT ESC 6, DATA [SI] [DI] si l'on veut que le diviseur soit la quantité située à l'adresse DATA + (SI) + (DI).

HLT — Halte

Permet la synchronisation du microprocesseur sur un événement extérieur, car ce dernier ne repart que sur un « reset » ou après traitement d'une interruption.

Code machine

1111 0100			

Durée

8086	8088	186, 188	286
2	2	2	2

Flags : Affectés aucun
Indéfinis

Exemples :

HLT codée F4

IDIV — Division signée

Réalise la division signée d'un nombre de 16 bits, contenu dans AX (ou 32 bits contenu dans DX et AX), par un nombre de 8 bits, indiqué par le deuxième opérande. Le quotient est stocké dans AL (ou AX) et le reste dans AH (ou DX).

La valeur **absolue** maximale du quotient étant 7F (ou 7FFF), si cette valeur est dépassée, il y a génération d'une interruption du type 0. Les flags (CS) et (IP) sont stockés en pile, (CS) prend la valeur rangée en 2H (16 bits) et (IP) celle rangée en 0H (16 bits). (SP) est décrémenté de 6. Les flags IF et TF sont mis à 0.

Le deuxième opérande (diviseur) peut être le contenu d'un registre ou d'une case mémoire.

Code machine

1111 011w	mod 111 r/m		

w = 0 DIVIDENDE = (AX)
QUOTIENT = (AL)
RESTE = (AH)

w = 1 DIVIDENDE = (DX,AX)
QUOTIENT = (AX)
RESTE = (DX)

Durée

	8086	8088	186, 188	286
registre 8 bits	101 – 112	101 – 112	44 – 52	17
mémoire 8 bits	(107 – 118) + AE	(107 – 118) + AE	50 – 58	20
registre 16 bits	165 – 184	165 – 184	53 – 61	25
mémoire 16 bits	(171-190) + AE	(175 – 194) + AE	59 – 67, 63 – 71	28

Flags : Affectés tous mais non significatifs
Indéfinis

Exemples :

IDIV BL codée F6 FB
IDIV WORD_BETA codée F7 selon WORD_BETA

Notes :

1. Avant une division, il faut utiliser soit CBW (8 en 16 bits), soit CWD (16 en 32 bits)

2. Dans les divisions signées, la convention est que le *reste* doit être du *même signe* que le *dividende*.

Ainsi, dans la division de ±26 par ±7, nous aurons (après conversion en décimal) :

$$-26 = (-3) \times 7 - 5$$
$$-26 = 3 \times (-7) - 5$$
$$26 = 3 \times 7 + 5$$
$$26 = (-3) \times (-7) + 5$$

ce qui signifie que la division est d'abord *non* signée : $26 = 3 \times 7 + 5$ puis les signes sont mis : au reste en fonction de celui du dividende, au quotient en fonction de ceux du diviseur et du dividende.

IMUL — Multiplication signée

Le contenu de l'accumulateur AL (ou AX) est multiplié par le contenu du deuxième opérande. Le résultat est stocké, signé, dans AX (ou DX et AX).
Si le résultat est étendu à 16 ou 32 bits, le carry est mis à 0 ; sinon, il est mis à 1 ainsi que OF.

Code machine

1111 011w	mod 101 r/m		

w = 0 Multiplicande = (AL)
Résultat = (AX), extension (AH)

w = 1 Multiplicande = (AX)
Résultat = (DX,AX), extension (DX)

Durée

	8086	8088	186, 188	286
registre 8 bits	80–98	80–98	25–28	13
mémoire 8 bits	(86–104)+AE	(86–104)+AE	31–34	16
registre 16 bits	128–154	128–154	34–37	21
mémoire 16 bits	(134–160)+AE	(138–164)+AE	40–43	24

Flags : Affectés CF, OF
Indéfinis AF, PF, ZF, SF

Exemples :

IMUL BL codée F6 EB

IMUL WORD_BETA codée F7 selon WORD_BETA

Note : si le multiplicateur est de 16 bits, il faut étendre le contenu de AL à 16 bits par CBW.

IMUL immédiat

Cette opération, propre aux iAPX 186/188/286 permet la multiplication du contenu d'un registre ou d'une case mémoire par une **donnée** (8 ou 16 bits) avec stockage du résultat dans un registre qui peut être différent de celui contenant le multiplicande.

Le **résultat** doit être **au maximum de 16 bits** signés. Le carry et l'overflow sont alors mis à 0, sinon ils sont mis à 1.

Code machine

0110 10s1	mod reg r/m	donnée	donnée si s=0

si s=1, la donnée est de 8 bits (ne pas confondre avec w)

Durée

	8086	8088	186, 188	286
registre à registre	—	—	22-25	21
mémoire à registre	—	—	29–32	24

Flags : Affectés CF, OF
Indéfinis AF, PF, ZF, SF

Exemples :

IMUL BX,SI,5 codée 69 DE 05

range le contenu de SI multiplié par 5 dans BX.

IMUL BX,[SI],5 codée 6B IC 05

range le contenu de la case mémoire (16 bits) pointée par (SI), par rapport à (DS), multiplié par 5, dans BX.

IN — Entrée dans l'accumulateur

Charge un octet (ou deux) dans l'accumulateur à partir d'un port.
— Le port peut être défini par le deuxième octet de l'instruction :

Code machine

1110 010w	port		

w = 0 on charge 8 bits dans AL
w = 1 on charge 16 bits dans AX

Durée

8086	8088	186, 188	286
10	14	10	5

Flags : Affectés aucun
Indéfinis

Exemples :

IN AX, WORD_PORT codée E5 N
IN AL, BYTE_PORT codée E4 N
(N = Numéro du port WORD_PORT ou BYTE_PORT)

Note : il ne peut y avoir que 256 *ports* ainsi adressés.

— Le port peut être défini par le contenu de DX.

Code machine

1110 110w			

w=0 on charge 8 bits dans AL depuis ((DX))

w=1 on charge 16 bits dans AX depuis ((DX)) et ((DX)+1)

Durée

8086	8088	186, 188	286
8	12	8	5

Flags : Affectés aucun
Indéfinis

Exemples :

IN AX,DX codée ED
IN AL,DX codée EC

Note : il peut y avoir 64 K ports de 8 bits.

INC — Incrémentation

Ajoute 1 au contenu de l'opérande. **Il n'y a pas de carry généré.**
— L'opérande peut être un registre (16 bits).

0100 0 reg			

Durée

8086	8088	186, 188	286
4	3	3	2

Flags : Affectés AF, OF, PF, SF, ZF
Indéfinis

Exemples :

INC AX codée 40
INC DI codée 47

— L'opérande peut être un registre (8 ou 16 bits) ou une case mémoire (ou deux) :

1111 111w	mod 000 r/m		

w=0 : 8 bits

Durée

	8086	8088	186, 188	286
registre	3	3	3	2
mémoire	15+AE	23+AE	15	7

Flags : Affectés AF, OF, PF, SF, ZF
Indéfinis

Exemples :

INC AL codée FE C0

INC WORD_BETA codée FF code BETA

INC WORD PTR [SI] codée FF 04

PTR permet de préciser que l'on désire incrémenter le mot (WORD) de 16 bits situé à l'adresse (SI) par rapport à (DS).

INS — Chargement mémoire depuis un port

Cette instruction permet le chargement d'un mot (8 ou 16 bits) dans une case mémoire d'adresse définie par le contenu de DI par rapport à l'extra-segment (ES) depuis un port d'adresse défini par le contenu de DX.

Après le transfert, le contenu de DI est automatiquement incrémenté de 1 (ou 2), si le flag de direction (DF) est nul (l'instruction CLD a été exécutée).
Cette instruction peut être répétée n fois si elle est précédée du préfixe REP.

0110 110w			

w=0, le mot est de 8 bits

Durée

	8086	8088	186, 188	286
une fois	—	—	14	5
n fois	—	—	8+8*n	5+4*n

Flags : Affectés aucun
Indéfinis

Exemples :

INS TAB_CONS, DX
si TAB_CONS a été définie comme un ensemble de consignes de 8 bits pointées par DI, par rapport à ES.

INT — Interruption

Comme RST x pour le 8085, cette instruction est équivalente à un appel long, mais les nouvelles valeurs de (CS) et de (IP) sont stockées à des adresses définies par construction.

Cette instruction provoque le stockage des flags dans la pile, la mise à zéro des flags TF et IF, la sauvegarde des contenus de CS et de IP. Les nouveaux contenus sont chargés à partir des cases mémoires dont les adresses dépendent du « *type* » de l'interruption, le contenu de IP étant chargé le dernier.
Le pointeur de pile est décrémenté de 6.

Il y a 256 types donc 1 K octets sont « *réservés* » aux interruptions ! (voir chapitre précédent).

Code machine

1100 110v	type si v=1		

si v=0 INT de type 3

Durée

	8086	8088	186, 188	286
type ≠ 3	52	71	47	23+m*
type 3	51	72	45	23+m

*m = nombre d'octets de l'instruction suivante

Flags : Affectés IF=0, TF=0
Indéfinis

Exemples :

1) INT 3 codée CC

Le contenu de IP sera celui des cases mémoires d'adresses 0CH et 0DH (C + 1) − (4 × 3 et 4 × 3 + 1) ; le contenu de CS sera celui des cases mémoires d'adresse 0FH et 0FH (E + 1) − (4 × 3 + 2 et (4 × 3 + 2) + 1).

2) INT 67 codée CD 43

Le contenu de IP sera celui des cases mémoires d'adresse 010CH (4 × 67) et 010DH ; celui de CS sera le contenu des cases mémoires 010EH et 010FH.

INTO — Interruption sur overflow

Cette interruption est du type 4 mais il s'agit d'une véritable interruption car l'instruction n'est effective que si le flag OF (overflow) vaut 1.

Code machine

1100 1110			

Durée

	8086	8088	186, 188	286
test seul	4	4	4	3
effective	53	73	48	23 + m*

*m est le nombre d'octets de l'instruction suivante

Flags : Affectés IF = 0, TF = 0
Indéfinis

Exemples :

INTO codée CE

si OF = 1, le contenu de CS sera celui des cases mémoires d'adresse 12H et 13H et celui de IP sera celui des cases mémoires d'adresses 10H et 11H (type 4 donc 4 × 4 = 16 = 10H). Le pointeur de pile sera décrémenté de 6.

IRET — Retour après interruption

Cette instruction termine un sous-programme de traitement d'une interruption.
Les contenus de IP,CS et des flags sont rechargés depuis la pile. Le pointeur de pile est incrémenté de 6.

Code machine

1100 1111			

Durée

8086	8088	186, 188	286
32	44	28	17+m*

*m = nombre d'octets de l'instruction suivante

Flags : Affectés tous (restauration des flags)
Indéfinis

Exemples :

IRET codée CF

JA - JNBE — Saut si supérieur à, saut si non inférieur, ni égal à (arithmétique non signée). ***A = Above, B = Below.***

Il s'agit d'un saut relatif, à l'intérieur du segment, de +127 à −128 octets, à partir de l'adresse de l'instruction qui **suit** JA ou JNBE **si les flags CF et ZF sont nuls.**

Code machine

0111 0111	dec		

dec = nombre d'octets, signé, de +127 (7FH) à −128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7+m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

```
JA   SUP                codée 77 dec
JNBE N_INF_NI_EG        codée 77 dec
```

Note : cette instruction suit généralement une instruction de comparaison (CMP) entre nombres **non signés.**

JAE - JNB — Saut si supérieur ou égal à, ou saut si non inférieur à (arithmétique non signée) ***A = Above, B = Below.***

Il s'agit d'un saut relatif, à l'intérieur du segment, de + 127 à – 128 octets, à partir de l'instruction qui suit JAE ou JNB si le flag CF est nul.

Code machine

0111 0011	dec		

dec = nombre d'octets, signé, de + 127 (7FH) à – 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7 + m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JAE SUP_EGAL codée 73 dec
JNB NON_INF codée 73 dec

Note : cette instruction suit généralement une instruction de comparaison (CMP) entre nombres **non signés**.

JB - JNAE — Saut si inférieur à, ou saut si non supérieur, ni égal à (arithmétique non signée) ***A = Above, B = Below.***

Il s'agit d'un saut relatif, à l'intérieur du segment, de + 127 à – 128 octets, à partir de l'adresse de l'instruction qui **suit** JB ou JNAE **si le flag CF vaut 1.**

Code machine

0111 0010	dec		

dec = nombre d'octets, signé, de + 127 (7FH) à – 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7+m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

```
JB    INF              codée 72 dec
JNAE N_EG_N_SUP        codée 72 dec
```

Note : cette instruction suit généralement une instruction de comparaison (CMP) entre nombre **non signés.**

JBE - JNA — Saut si inférieur ou égal à, ou saut si non supérieur à (arithmétique non signée). ***B = Below, A = Above.***

Il s'agit d'un saut relatif, à l'intérieur du segment de + 127 à − 128 octets, à partir de l'adresse de l'instruction qui **suit** JBE ou JNA, **si l'un des flags CF ou ZF est égal à 1** (CF = 1 : inférieur à, ZF = 1 : égal à).

Code machine

0111 0110	dec		

dec : nombre d'octets, signé, de + 127 (7FH) à − 128 (80H)

Durée

	8086	8088	186, 188	286
sauf effectif	16	16	13	7+m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JBE INF_EGAL codée 76 dec
JNA NON_SUP codée 76 dec

Note : cette instruction suit généralement une instruction de comparaison (CMP) entre deux nombres **non signés.**

JC — Saut si Carry = 1

Il s'agit d'un saut relatif, à l'intérieur du segment, de + 127 à – 128 octets, à partir de l'adresse qui **suit JC si le flag CF vaut 1** (l'instruction est identique dans son principe à JB ou JNAE mais plus explicite en assembleur : utilisée après les décalages/rotations).

Code machine

0111 0010	dec		

dec : nombre d'octets, signé, de + 127 (7FH) à – 128 (80H)

Durée

	8086	8088	186, 188	286
sauf effectif	16	16	13	7 + m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JC DEP codée 72 dec

JCXZ — Saut si le contenu de CX est nul

Il s'agit d'un saut relatif à l'intérieur du segment, de + 127 à – 128 octets, à partir de l'adresse de l'instruction qui **suit** JCXZ si le **contenu du registre CX est nul.**

Code machine

1110 0011	dec		

dec : nombre d'octets, signé, de + 127 (7FH) à – 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	18	18	16	8+m*
pas de saut	6	6	5	4

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JCXZ SUITE codée E3 dec

JE - JZ — Saut si égal, ou saut si ZF = 1

Il s'agit d'un saut relatif, à l'intérieur du segment, de + 127 à − 128 octets, à partir de l'adresse de l'instruction qui **suit** JE ou JZ **si le flag** ZF **vaut 1.**

Code machine

0111 0100	dec		

dec : nombre d'octets, signé, de + 127 (7FH) à − 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7+m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

1) CMP CX,DX codée 39 CA
JE TROUVE codée 74 dec

le saut n'a lieu que si (CX) = (DX)

2) SUB AX,BX codée 2B C3
JZ EXACT codée 74 dec

le saut n'a lieu que si (AX) = (BX).

JG - JNLE — Saut si plus grand que, ou saut si pas plus petit que ni égal à (arithmétique signée) ***G = Greater, L = Less.***

Il s'agit d'un saut relatif à l'intérieur du segment, de + 127 à − 128 octets, à partir de l'adresse de l'instruction qui **suit** JG ou JNLE, **si les flags SF et OF sont égaux** (1 ou 0) **et** si le flag **ZF est nul** ((SF ⊕ OF) + ZF = 0).

Code machine

0111 1111	dec		

dec : nombre d'octets, signé, de + 127 (7FH) à − 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7 + m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

```
JG   PLUS_GRAND          codée 7F dec
JNLE N_PLUS_PTI_NI_EG    codée 7F dec
```

Note : cette instruction suit généralement une instruction de comparaison (CMP) entre 2 **nombres signés.**

JGE - JNL — Saut si pas plus petit que ou saut si plus grand que ou égal à (arithmétique signée) ***L = Less, G = Greater.***

Il s'agit là d'un saut relatif, à l'intérieur du segment, de + 127 à − 128 octets, à partir de l'adresse de l'instruction qui **suit** JNL ou JGE **si les flags SF et OF sont égaux** (1 ou 0) : (SF ⊕ OF = 0).

Code machine

0111 1101	dec		

dec : nombre d'octets, signé, de + 127 (7FH) à − 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7+m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JNL NON_PL_PTI codée 7D dec
JGE PL_GRAND_EGAL codée 7D dec

Note : cette instruction suit généralement une instruction de comparaison (CMP) entre 2 **nombres signés.**

JL - JNGE — Saut si plus petit que ou saut si pas plus grand que ni égal à (arithmétique signée) ***L = Less, G = Greater.***

Il s'agit d'un saut relatif, à l'intérieur d'un segment, de +127 à −128 octets, à partir de l'adresse de l'instruction qui **suit** JL ou JNGE si **les flags SF et OF ne sont pas égaux** (SF ⊕ OF = 1).

Code machine

0111 1100	dec		

dec : nombre d'octets, signé, de +127 (7FH) à −128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7+m*
pas de saut	4	4	4	3

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JL	PLUS_PTI	codée 7C dec
JNGE	P_PL_GRD_NI_EG	codée 7C dec

Note : cette instruction suit généralement une instruction de comparaison (CMP) **entre nombres signés.**

JLE-JNG — Saut si plus petit ou égal à, ou saut si pas plus grand que (arithmétique signée) ***L = Less, G = Greater.***
Il s'agit d'un saut relatif, à l'intérieur du segment, de + 127 à − 128 octets à partir de l'adresse de l'instruction qui **suit** JLE ou JNG **si le flag ZF vaut 1 ou si les flags SF et OF ne sont pas égaux** ((SF ⊕ OF) + ZF = 1).

Code machine

0111 1110	dec		

dec : nombre d'octets, signé, de + 127 (7FH) à − 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7 + m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JLE	PL_PTI_EG	codée 7E dec
JNG	PAS_PL_GRD	codée 7E dec

Note : cette instruction suit généralement une instruction de comparaison (CMP) entre **deux nombres signés.**

JMP — Saut inconditionné

Le saut inconditionné peut avoir lieu à l'intérieur d'un segment, sur une étendue de 256 octets ou dans la totalité du segment, ou à l'extérieur du segment.

Dans le premier cas, on ne définira que l'étendue du saut relatif de +127 à −128 ou de +32767 (7FFFH) à −32768 (8000H) octets.
Dans le deuxième cas, on donnera les nouvelles valeurs des contenus de CS et de IP.

Dans les deux cas, les données peuvent être immédiates (contenues dans l'instruction) — saut direct — ou définies par le contenu de cases mémoire ou registre — saut indirect.

1. Saut intrasegment, direct, *relatif*

1-1 **Long** (+32767 à −32768 octets)

Code machine

1110 1001	dec. bas	dec. haut	

Durée

8086	8088	186, 188	286
15	15	13	7+m*

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JMP NEAR_LABEL codée E9 dec bas dec haut

1-2 **Court** (+127 à −128 octets)

Code machine

1110 1011	dec		

Durée

8086	8088	186, 188	286
15	15	13	7+m*

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JMP SHORT NEAR_LABEL   codée EB dec

Remarque : l'assembleur ASM86 ajoute un NOP (codé 90) à cette instruction si l'adresse de destination est plus haute et que l'on n'a pas écrit SHORT car à la première passe, il prévoit une instruction de trois octets.

2. Saut intra-segment, indirect

Ici, la nouvelle valeur du contenu de IP est le contenu d'une case mémoire ou d'un registre.

Code machine

1111 1111	mod 100 r/m		

Durée

	8086	8088	186, 188	286
registre	11	11	11	7+m*
mémoire	18+AE	18+AE	17	11+m

*m : nombre d'octets de l'instruction suivante

Flags : Affectés   aucun
Indéfinis

Exemples :

JMP AX   ; (IP) = (AX)   codée FF E0
JMP TABLE [BX]   codée FF selon TABLE

Note :
— ne pas confondre JMP [BX] avec JMP BX ! codées respectivement FF 27 et FF E3, la première charge IP à partir du contenu de la case mémoire pointée par (BX) par rapport à (DS) la deuxième charge IP au contenu de BX ;

— en ASSEMBLEUR on écrit JMP WORD PTR [BX] pour préciser qu'il s'agit d'un saut intra-segment (seul IP est chargé).

3. Saut inter-segment

3-1 **direct**

Code machine

1110 1010	IPB	IPH	CSB	CSH

(IP) = IPH, IPB
(CS) = CSH, CSB

Durée

8086	8088	186, 188	286
15	15	13	11+m*

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

```
JMP addr.                addr sur 4 octets
JMP FAR__LABEL
```

3-2 **indirect**

Code machine

1111 1111	mod 101 r/m		

mod ≠ 11

(CS) = (AE + 2)
(IP) = (AE)

Durée

8086	8088	186, 188	286
24+AE	24+AE	26	15+m*

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

```
JMP DWORD PTR [BP] [DI]
```

Note : on ne peut évidemment pas utiliser le contenu d'un registre.

JNA — voir JBE
JNAE — voir JB
JNB — voir JAE
JNBE — voir JA
JNG — voir JLE
JNGE — voir JL
JNL — voir JGE
JNLE — voir JG

JNC — Saut s'il n'y a pas de Carry (CF = 0)

Il s'agit d'un saut relatif, à l'intérieur du segment, de + 127 à – 128 octets, à partir de l'adresse de l'instruction qui suit JC si CF est nul.

Code machine

0111 0011	dec		

dec : nombre d'octets, signé, de + 127 (7FH) à – 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7 + m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JNC BIT__ZERO codée 73 dec

JNE - JNZ — Saut si différent de, ou saut si ZF = 0

Il s'agit d'un saut relatif, à l'intérieur du segment de + 127 à – 128 octets, à partir de l'adresse de l'instruction qui **suit** JNE ou JNZ **si le flag ZF est égal à 0.**

Code machine

0111 0101	dec		

dec : nombre d'octets signés, de + 127 (7FH) à – 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7 + m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

1)	CMP AL, '['	codée 3C 5B
	JNE ERREUR	codée 75 dec
2)	DEC CL	codée FE C9
	JNZ BOUCLE	codée 75 dec

JNO — Saut s'il n'y a pas d'overflow

Il s'agit d'un saut relatif, à l'intérieur du segment, de + 127 à − 128 octets, à partir de l'adresse de l'instruction qui suit JNO **si le flag OF est nul.**

Code machine

0111 0001	dec		

dec : nombre d'octets, signé, de + 127 (7FH) à − 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7+m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JNO LIMIT	codée 71 dec

JNP - JPO — Saut si pas de parité ou saut si parité impaire ***(O = odd)***

Il s'agit d'un saut relatif, à l'intérieur du segment, de + 127 à − 128 octets, à partir de l'adresse de l'instruction qui suit JNP ou JPO **si le flag PF est nul.**

Code machine

0110 1011	dec		

dec : nombre d'octets, signé, de + 127 (7FH) à − 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif pas de saut	16 4	16 4	13 4	7 + m* 3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

1) JNP ERREUR codée 7B dec
2) JPO ERREUR codée 7B dec

Note : la *parité* d'une donnée est *celle du nombre de 1.*

JNS — Saut si positif, ou saut si le flag SF est nul

Il s'agit d'un saut relatif, à l'intérieur du segment, de − 127 à − 128 octets, à partir de l'adresse de l'instruction qui **suit** JNS **si le flag** SF est **nul.**

Code machine

0111 1001	dec		

dec : nombre d'octet, signé, de + 127 (7FH) à + 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif pas de saut	16 4	16 4	13 4	7 + m* 3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JNS POSITIF codée 79 dec

JNZ — voir JNE

JO — Saut si le flag OF vaut 1

Il s'agit d'un saut relatif à l'intérieur du segment, de +127 à −128 octets, à partir de l'adresse de l'instruction qui **suit** JO, **si le flag OF=1.**

Code machine

0111 0000	dec		

dec : nombre d'octets signé, de +127 (7FH) à −128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7+m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JO DEPAS codée 70 dec

JP - JPE — Saut si la parité existe, ou saut si la parité est paire *E=Even*

Il s'agit d'un saut relatif à l'intérieur du segment de +127 à −128 octets, à partir de l'adresse de l'instruction qui **suit** JP ou JPE **si le flag PF vaut 1.**

Code machine

0111 1010	dec		

dec : nombre d'octets, signé, de +127 (7FH) à −128 (80H)

Durée

	8086	8088	186, 188	286
sauf effectif	16	16	13	7+m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

1) JP ERREUR codée 7A dec

2) JPE ERREUR codée 7A dec

Note : la *parité* d'une donnée est *celle du nombre de 1.*

JPE — voir JP

JPO — voir JNP

JS — Saut si négatif ou si le flag SF vaut 1

Il s'agit d'un saut relatif, à lintérieur du segment, de + 127 à − 128 octets à partir de l'adresse de l'instruction qui **suit JS si le flag SF vaut 1.**

Code machine

0111 1000	dec		

dec : nombre d'octets signé, de + 127 (7FH) à − 128 (80H)

Durée

	8086	8088	186, 188	286
saut effectif	16	16	13	7 + m*
pas de saut	4	4	4	3

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

JS NEGATIF codée 78 dec

Table des différents sauts conditionnés

Mnémoniques	Codes	Conditions	
Tests non signés (supérieur/inférieur)			
JA/JNBE	77	$(CF + ZF) = 0$	>
JAE/JNB	73	$CF = 0$	≥
JB/JNAE	72	$CF = 1$	<
JBE/JNA	76	$(CF + ZF) = 1$	≤
JC	72	$CF = 1$	
JE/JZ	74	$ZF = 1$	=
JNC	73	$CF = 0$	
JNE/JNZ	75	$ZF = 0$	≠
JNP/JPO	7B	$PF = 0$	
JP/JPE	7A	$PF = 1$	
Tests signés (plus grand que/plus petit que)			
JG/JNLE	7F	$((SF \oplus OF) + ZF) = 0$	>
JGE/JNL	7D	$(SF \oplus OF) = 0$	≥
JL/JNGE	7C	$(SF \oplus OF) = 1$	<
JLE/JNG	7E	$((SF \oplus OF) + ZF) = 1$	≤
JNO	71	$OF = 0$	
JNS	79	$SF = 0$	>0
JO	70	$OF = 1$	
JS	78	$SF = 1$	<0

Notes : + ⟶ ou :
⊕ ⟶ ou exclusif

LAHF — Chargement de AH avec les flags

Les flags SF, ZF, AF, PF, CF (ceux des 8080, 8085) sont mis dans le registre AH selon le schéma suivant (X indéfini) :

SF	ZF	X	AF	X	PF	X	CF

Code machine

1001 1111			

Durée

8086	8088	186, 188	286
4	4	2	2

Flags : Affectés aucun
Indéfinis

Exemples :

LAHF codée 9F

Permet le test de AF à l'intérieur de AH par masque ou rotations.

LDS — Chargement simultané d'un registre et du segment des données DS

Le contenu du registre précisé est remplacé par le mot situé à l'adresse définie (REG)←(AE), celui de DS est remplacé par le mot suivant (DS)←(AE+2).

Code machine

1100 0101	mod reg r/m		

mod ≠ 11

Durée

8086	8088	186, 188	286
16+AE	24+AE	18, 26	7

Flags : Affectés aucun
Indéfinis

Exemples :

– LDS BX,TABLE [SI] codée C5 selon code TABLE

On aura, après exécution :
(BX)←(DS : TABLE+(SI))
(DS)←(DS : TABLE+(SI)+2)

DS : signifie par rapport à (DS) la quantité qui suit étant l'adresse effective dans le segment DS.

– LDS SI,SOURCE codée C5 selon code SOURCE

permet de charger (DS) et (SI) avant une opération sur chaîne de mots, l'adressage avec SI étant référencé par rapport à (DS).

LEA — Chargement d'une adresse effective

Le contenu du registre spécifié est remplacé par la *valeur* de l'adresse effective.

(REG)←AE

Code machine

1000 1101	mod reg r/m		

mod ≠ 11

Durée

8086	8088	186, 188	286
2+AE	2+AE	6	3

Flags : Affectés aucun
Indéfinis

Exemples :

LEA BX, [BP] [DI] codée 8D 1B

Après exécution, nous aurons :

(BX)←(BP) + (DI)

(DS) n'intervient pas.

— On utilise LEA BX, TAB_ASCII pour charger BX avec l'adresse effective de la TABLE ASCII avant d'employer XLAT (voir cette instruction), les codes ASCII étant définis comme « variables ».

LEAVE — Sortir d'une procédure

Cette instruction termine une procédure dans laquelle la première instruction est ENTER.
Elle précède l'instruction de retour : RET.
Le contenu de BP est copié dans SP pour libérer la zone mémoire réservée par ENTER, puis l'ancienne valeur du contenu de BP est restaurée par un POP BP.

Code machine

1100 1001			

Durée

8086	8088	186, 188	286
—	—	8	5

Flags : Affectés aucun
Indéfinis

Exemples :

LEAVE codée C9

LES — Chargement simultané d'un registre et de l'extra-segment ES

Le contenu du registre précisé est remplacé par le mot situé à l'adresse définie (REG)←(AE). Celui de ES est remplacé par le mot suivant (ES)←(AE + 2).

Code machine

1100 0100	mod reg r/m		

mod ≠ 11

Durée

8086	8088	186, 188	286
16 + AE	24 + AE	18, 26	7

Flags : Affectés aucun
Indéfinis

Exemples :

— LES BX, TABLE [SI] codée C4 selon code TABLE

on aura, après exécution :

(BX)←(DS : TABLE + (SI))
(ES)←(DS : TABLE + (SI) + 2)

— LES DI, DESTINATION codée C4 selon code DESTINATION

permet de charger (ES) et (DI) avant une opération sur chaîne de mots, l'adressage avec DI étant référencé par rapport à (ES) — (voir LDS).

LOCK — Commande la mise à zéro du signal lock

Cette instruction est utilisée en multiprocessing pour éviter les conflits, en particulier lors d'accès mémoire. Il s'agit d'un *préfixe*.

La sortie LOCK est mise à zéro pendant la durée de l'instruction qui suit.

Code machine

1111 0000			

Durée

8086	8088	186, 188	286
2	2	2	0

Flags : Affectés aucun
Indéfinis

Exemples :

LOCK codée F0

Note : pour un fonctionnement correct en cas de multiprocessing, il faut assurer au microprocesseur prioritaire (premier demandeur), un usage exclusif du bus, à l'aide d'un préfixe LOCK et d'un peu de logique câblée.

LODS — Chargement de l'accumulateur à partir d'une suite de données

Le contenu de l'accumulateur AL (ou AX) est remplacé par le mot d'un (ou deux) octet dont l'adresse est contenue dans SI — par rapport à (DS).
Le contenu de SI est incrémenté de 1 (ou 2) ou décrémenté de 1 (ou 2) suivant que le flag DF est nul ou non.
Cette instruction peut, éventuellement, être répétée n fois si elle est précédée du préfixe REP.

Code machine

1010 110w			

w = 1 : chargement de AX
w = 0 : chargement de AL

Durée

	8086	8088	186, 188	286
une opération	12	16	12	5
n opérations	9+13*n	9+17*n	6+11*n	5+4*n

Flags : Affectés aucun
Indéfinis

Exemples :

1)
```
CLD                        ; DF=0
                           ; on incrémentera (SI) de 1
MOV  SI, BYTE__TABLE
LODS BYTE__TABLE           codée AC
```

2)
```
STD                        ; DF=1
                           ; on décrémentera (SI) de 2
MOV  SI, WORD__TABLE
LODS WORD__TABLE
```

LOOP — Boucle, opérations répétées tant que le compteur n'est pas à zéro

Cette instruction décrémente le contenu du **registre CX** et provoque un saut **relatif**, court si ce dernier n'est pas nul.

Code machine

1110 0010	dec		

dec : nombre d'octets, signé, de +127 (7FH) à −128 (80H).

Durée

	8086	8088	186, 188	286
saut effectif	17	17	16	8+m*
pas de saut	5	5	6	4

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples : Somme de N nombres de 2 octets commençant en TABLE

```
         MOV CX, N
         MOV AX, 0
         MOV SI, 0
SUITE :  ADD AX, TABLE [SI]
         ADD SI, 2
         LOOP SUITE              codée E2 selon nombre d'oc-
                                 tets du code TABLE
```

LOOP SUITE est identique à

```
DEC CX
JNZ SUITE
```

LOOPE-LOOPZ — Boucle si égal à, ou boucle si ZF = 1

Cette instruction décrémente le contenu de CX et provoque un **saut relatif**, court, si le flag **ZF vaut 1 ET si le contenu de CX n'est pas nul** (on sort de la boucle si ZF = 0 ou si (CX) = 0).

Code machine

1110 0001	dec		

dec : nombre d'octets, signé, de + 127 (7 FH) à − 128 (80 H)

Durée

	8086	8088	186, 188	286
sauf effectif	18	18	16	8 + m*
pas de saut	6	6	6	4

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :
Recherche du premier terme **non nul** dans une suite de N octets, commençant en TABLE.

```
          MOV CX, N
          MOV SI, -1
SUITE :   INC SI
          CMP TABLE [SI], 0
          LOOPE SUITE          ;codée E1 selon code de TABLE
          JNE TROUVE
          ...                  ;ici la table ne contient que des
                                zéros
TROUVE : ...                   ;(SI) pointe le 1er octet non nul
```

LOOPNE-LOOPNZ — Boucle si différent de, ou boucle si le flag ZF = 0

Cette instruction décrémente le contenu de CX, et provoque un **saut relatif**, court, **si le flag ZF est nul, ET si le contenu de CX n'est pas nul,**(on sort de la boucle si ZF = 1 ou si (CX) = 0).

Code machine

1110 0000	dec		

dec : nombre d'octets, signé, de + 127 (7 FH) à − 128 (80 H)

Durée

	8086	8088	186, 188	286
sauf effectif	19	19	16	8 + m*
pas de saut	5	5	5	4

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples : recherche du 1er terme *nul* dans une chaîne de N octets, commençant en TABLE.

```
          MOV CX, N
          MOV SI, -1
SUITE :   INC SI
          CMP TABLE [SI],0
          LOOPNE SUITE
          JE TROUVE
          ...                  ;ZF = 0 donc (CX) = 0
TROUVE : ...                   ;SI () pointe l'octet nul
```

Remarque : pour LOOPZ ou LOOPNZ on se branche vers les instructions traitant la **condition recherchée** par JNZ ou JZ (LOOPE et JNE, LOOPNE et JE).

LOOPNZ — voir LOOPNE

LOOPZ — voir LOOPE

MOV — Transfert de données

Cette instruction présente 7 cas selon la nature des opérandes.

1. Vers une case mémoire depuis l'accumulateur (AL ou AX)

Code machine

1010 001w	ad. basse	ad. haute	

ad. : adresse w = 1 : AX, w = 0 : AL

Durée

8086	8088	186, 188	286
10	14	9	3

Flags : Affectés aucun
Indéfinis

Exemples :

MOV [1064 H], AL codée A2 64 10
MOV [1064 H], AX codée A3 64 10

2. Vers l'accumulateur depuis une case mémoire

Code machine

1010 000w	ad. basse	ad. haute	

ad : adresse, w = 1 : AX, w = 0 : AL

Durée

8086	8088	186, 188	286
10	14	9	5

Flags : Affectés aucun
Indéfinis

Exemples :

MOV AL, [1064 H] codée A0 64 10
MOV AX, [1064 H] codée A1 64 10

3. Transfert entre registre ou registre et case mémoire

Code machine

1000 10dw	mod reg r/m		

d = 1 le registre, ou le premier nommé, est la destination et est défini par *reg*

Durée

	8086	8088	186, 188	286
entre registres	2	2	2	3
vers un registre (d = 1)	8 + AE	12 + AE	12	5
depuis un registre (d = 0)	9 + AE	13 + AE	9	3

Flags : Affectés aucun
Indéfinis

Exemples :

MOV BX, AX codée 8B D8
MOV CX, [SI] codée 8B 0C
MOV [SI], CX codée 89 0C

4. Chargement d'une donnée dans un registre

Code machine

1011 wreg	donnée	donnée si w = 1	

Durée

8086	8088	186, 188	286
4	4	3-4	2

Flags : Affectés aucun
Indéfinis

Exemples :

MOV AX, 77H codée B4 77 00

5. Chargement d'une donnée dans une case mémoire ou un registre

Code machine

1100 011w	mod 000 r/m	donnée	donnée si w = 1

Durée

	8086	8088	186, 188	286
registre	4	4	4	2
mémoire	10+AE	14+AE	12−13	3

Flags : Affectés aucun
Indéfinis

Exemples :

MOV BX, 8FH codée C7 C3 8F 00
MOV WORD PTR [DI], 0C0F2H codée C7 05 F2 C0

Note : en ASSEMBLEUR, il faut préciser la taille du mot chargé en mémoire à l'aide de PTR.

6. Chargement d'un registre de segment depuis un registre ou une case mémoire

Code machine

1000 1110	mod 0 reg r/m		

reg ≠ 01

Durée

	8086	8088	186, 188	286
depuis un registre	2	2	2	2
depuis une case mémoire	8+AE	12+AE	9, 13	5

Flags : Affectés aucun
Indéfinis

EXTRAIT DE L'ASSEMBLEUR LIGNE A LIGNE ,POUR
SDK-86 ECRIT PAR L'AUTEUR.

(IP)	Mnémoniques		Codes
0000	MOV	BX,CX	@ 8B D9
0002	MOV	BX,[01234]	@ 8B 1E 34 12
0006	MOV	BX,[012]	@ 8B 1E 12 00
000A	MOV	BX,[01234][BP]	@ 8B 9E 34 12
000E	MOV	BX,[012][BP]	@ 8B 5E 12
0011	MOV	BX,[01234][BP][DI]	@ 8B 9B 34 12
0015	MOV	BX,[012][BP][DI]	@ 8B 5B 12
0018	MOV	BX,[BP][DI]	@ 8B 1B
001A	MOV	BX,CS:[01234][BP][DI]	@ 2E 8B 9B 34 12
001F	MOV	[01234],BX	@ 89 1E 34 12
0023	MOV	AL,[01234]	@ A0 34 12
0026	MOV	AH,[01234]	@ 8A 26 34 12
002A	MOV	AX,[01234]	@ A1 34 12
002D	MOV	CS:[01234][BP][DI]W,05678	@ 2E C7 83 34 12 78 56
0034	MOV	[01234]B,056	@ C6 06 34 12 56
0039	MOV	BL,045	@ B3 45
003B	MOV	BH,045	@ B7 45
003D	MOV	BX,04567	@ BB 67 45
0040	MOV	[01234],AX	@ A3 34 12
0043	MOV	DS,BX	@ 8E DB
0045	MOV	BX,DS	@ 8C DB
0047	MOV	DS,[DI]	@ 8E 1D
0049	MOV	[DI],DS	@ 8C 1D
004B	MOV	DS,CS:[DI]	@ 2E 8E 1D
004E			

Exemple de codages pour MOV.

Exemples :

MOV DS, AX codée 8E D8

Note : CS est interdit comme destination.
CS n'est modifié que par :
— un saut intersegment
— un appel intersegment
— une interruption

7. Chargement d'un registre ou d'une case mémoire depuis un registre de segment

Code machine

1000 1100	mod 0 reg r/m		

Durée

	8086	8088	186, 188	286
registre	2	2	2	2
mémoire	9+AE	13+AE	11, 15	5

Flags : Affectés aucun
Indéfinis

Exemples :

MOV DX, DS codée 8C D9

MOVS — Transfert de données

Transfère la donnée (octets ou mots) pointée par (SI) en une zone mémoire pointée par (DI). Après le transfert (SI) et (DI) sont incrémentés si DF est nul (l'instruction CLD a été exécutée), ou décrémentés si DF vaut un.

Grâce au préfixe REP (voir plus loin), cette instruction permet le transfert de bloc, même si les deux zones mémoires se recouvrent.

Code machine

1010 010w			

w=0 : transfert d'octets

Durée

	8086	8088	186, 188	286
une opération	18	26	9	5
n opérations	9+17*n	9+25*n	8+8*n	5+4*n

Flags : Affectés aucun
Indéfinis

Exemples :

MOV SI, TAB1

MOV DI, TAB2

MOV CX, NBR

REP MOVS TAB2, TAB1 codée A4 ou A5

Note : 1) (SI) pointe par rapport à (DS) et (DI) par rapport à (ES).

2) En ASSEMBLEUR, TAB1 et TAB2 ont été définies comme contenant des octets (DB) ou des mots (DW).

MUL — Multiplication non signée du contenu de l'accumulateur par celui d'un registre ou d'une case mémoire

Multiplie le contenu de l'accumulateur 8 bits (AL) ou 16 bits (AX) par le contenu d'un registre ou d'une case mémoire (8 ou 16 bits). Le résultat est dans AX ou AX et DX. Si l'octet (mot) de poids fort est nul, les flags CF et OF sont mis à zéro, sinon ils sont mis à 1. C'est-à-dire que si un octet multiplié par un octet donne (AH) = 0 ou si un mot (16 bits) multiplié par un mot donne (DX) = 0, les flags CF et OF sont mis à 0.

Code machine

1111 011 w	mod 100 r/m		

w = 0 multiplicande = (AL) résultat = (AX)

w = 1 multiplicande = (AX) résultat = (DX, AX)

Durée

	8086	8088	186, 188	286
8 bits : registre	70-77	70-77	26 – 28	13
mémoire	(76-83)	(76-83)	32 – 34	16
16 bits : registre	(118 – 133)	118 – 133	35 – 37	21
mémoire	(124 – 139)	(128 – 143)	41-43, 45-47	24

Flags : Affectés OF, CF
Indéfinis AF, PF, SF, ZF

Exemples :

1) MUL BL ou MUL AL, BL codée F6 E3
MUL BETA si BETA = BYTE codée F6 code BETA

2) MUL BX codée F7 E3
MUL ALPHA si ALPHA = WORD codée F7 code ALPHA

NEG — Complément à 2

Le contenu de l'opérande est complémenté à 2 (les uns deviennent des zéros et vice-versa et on ajoute 1).

Code machine

1111 011 w	mod 011 r/m		

Durée

	8086	8088	186, 188	286
registre	3	3	3	2
mémoire	16 + AE	24 + AE	3	2

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

1) si (AL) = 13H après NEG AL (codée F6 D8), nous avons (AL) = EDH (13 + ED = 00)

2) si (BETA) = AFH alors NEG BETA donne (BETA) = 51H

3) si (SI) = 2FC3H alors NEG SI (codée F7 DE) donne (SI) = D03DH

NOP — Pas d'opération

Code machine

10010000			

Durée

8086	8088	186, 188	286
3	3	3	3

Flags : Affectés aucun
Indéfinis

Exemples :

NOP

Note : NOP (codée 90H) peut être générée par ASM 86 comme 3ᵉ octet des sauts relatifs courts ou pour compléter une instruction qui a été prévue, à la première passe, plus longue qu'elle n'est en réalité.

NOT — Complément A 1

Le contenu de l'opérande est complémenté à 1, c'est-à-dire que les zéros deviennent des uns et vice-versa.

Code machine

1111 011w	mod 010 r/m		

Durée

	8086	8088	186, 188	286
registre	3	3	3	2
mémoire	16 + AE	24 + AE	10	7

Flags : Affectés aucun
Indéfinis

Exemples :

1) si (AL) = 13 H, alors NOT AL (codée F6 D0) donne (AL) = ECH (13 + EC = FF)

2) si (BETA) = AFH, alors NOT BETA donne (BETA) = 50H

3) si (SI) = 2FC3H, alors NOT SI (codée E7 D6) donne (SI) = D03CH

OR — OU (logique)

Réalise un *OU,* bit à bit, entre les contenus des deux opérandes, le résultat est *stocké* dans le *premier nommé.* Les flags CF et OF sont mis à zéro.

1. OU entre registre et case-mémoire

Code machine

0000 10dw	mod reg r/m		

d = 1 : le registre, le 1[er] nommé, est la destination et défini par reg

Durée

	8086	8088	186, 188	286
entre registre	3	3	3	2
mémoire-registre	16 + AE	24 + AE	10	7
registre-mémoire	9 + AE	13 + AE	10	7

Flags : Affectés CF = 0, OF = 0, PF, SF, ZF
Indéfinis AF

Exemples :

```
OR AH, BL       ; résultat dans AH   codée 0A E3
                ; (BL) inchangé
OR CX, DI                            codée 0B D7
OR AX, BETA     ; (BETA) = WORD
OR ALPHA, DX    ; (ALPHA) = WORD
```

2. OU immédiat avec l'accumulateur (AL ou AX)

Code machine

0000 110w	données	donnée si w = 1	

Durée

8086	8088	186, 188	286
4	4	3-4	3

Flags : Affectés SP, PF, ZF, CF=0, OF=0
Indéfinis AF

Exemples :

OR AL, 0F6 H codée 0C F6

OR AX, 400 H codée 0D 00 04

3. OU immédiat avec registre ou mémoire

Code machine

1000 000w	mod 001 r/m	donnée	donnée si w=1

Durée

	8086	8088	186, 188	286
registre	4	4	4	3
mémoire	17+AE	25+AE	16	7

Flags : Affectés SF, PF, ZF, CF=0, OF=0
Indéfinis AF

Exemples :

OR AH, 0F0H codée 80 CC F0
OR CL, 37H codée 80 C9 37
OR WORD PTR [SI], 0FF00H codée 81 0C 00 FF

OUT — Sortie d'un octet ou d'un mot

Le contenu de l'accumulateur est « sorti » par le port défini (0 à 255) ou par le port adressé via le contenu de DX (0 à 65535).

1) Port défini

Code machine

1110 011w	port		

Durée

8086	8088	186, 188	286
10	14	9	3

Flags : Affectés aucun
Indéfinis

Exemples :

OUT 44, AL codée E6 2C

OUT 250, AX codée E7 FA

2) Indirect

Code machine

1110 111w			

si w = 1 le port est (DX) + 1, (DX)

Durée

8086	8088	186, 188	286
8	12	7	3

Flags : Affectés aucun
Indéfinis

Exemples :

OUT DX, AL codée EE

OUT DX, AX codée EF

OUTS — Sortie du contenu mémoire par un port

Cette instruction permet de sortir le contenu d'une case mémoire (8 ou 16 bits) pointée par le contenu de SI par rapport au segment des données (DS) vers le port d'adresse défini par le contenu de DX.

Après le transfert, le contenu du registre SI est automatiquement incrémenté de 1 ou 2 si le flag de direction est nul (l'instruction CLD a été exécuté).
Cette instruction peut être répétée, n fois, si elle est précédée du préfixe REP.

Code machine

0110 111w			

w = 0 les mots sont de 8 bits

Durée

	8086	8088	186, 188	286
une opération	—	—	14	5
n opérations	—	—	8 + 8*n	5 + 4*n

Flags : Affectés aucun
Indéfinis

Exemples :

OUTS MESS,DX codée 6E

si MESS a été défini comme une suite de mots de 8 bits, constituant un message, pointés par le contenu de SI par rapport à DS.

POP — Restauration d'un mot depuis la pile

Le mot stocké (16 bits) en haut de la pile est rangé dans la destination, le contenu du registre SP est incrémenté de 2.

1. Rappel d'un contenu d'un registre

Code machine

0101 1 reg			

Durée

8086	8088	186, 188	286
8	12	10, 14	5

Flags : Affectés aucun
Indéfinis

Exemples :

POP DX codée 5A

2. Rappel du contenu d'un registre de segment

Code machine

000 reg 111			

Durée

8086	8088	186, 188	286
8	12	8, 12	20

Flags : Affectés aucun
Indéfinis

Exemples :

POP DS codée 1F

Note : POP CS est interdit.

3. Chargement d'une case mémoire depuis le haut de la pile

Code machine

1000 1111	mod 000 r/m		

Durée

	8086	8088	186, 188	286
mémoire	17+AE	25+AE	20, 24	5

Flags : Affectés aucun
Indéfinis

Exemples :

POP BETA [BX] codée 8F selon code BETA

Note : cette instruction peut servir à transmettre des paramètres. On peut évidemment coder, en machine, cette dernière instruction pour un registre mais elle nécessite deux octets.

POPA — Restauration de tous les registres (POP All)

Les mots de 16 bits stockés en haut de la pile sont chargés dans les registres généraux, pointeurs et index, DI étant chargé le premier et AX le dernier. Le contenu du pointeur de pile (SP) est décrémenté de 16D.

Code machine

0110 0001			

Durée

8086	8088	186, 188	286
—	—	51, 83	19

Flags : Affectés aucun
Indéfinis

Exemples :

POPA codée 61

Note : POPA est identique à la suite d'instructions :

POP DI
POP SI
POP BP
POP SP
POP BX
POP DX
POP CX
POP AX

POPF — Restauration des flags depuis le haut de la pile

Le haut de la pile, 16 bits, est chargé dans les flags, le contenu du registre SP est décrémenté de 2.

Code machine

1001 1101			

Durée

8086	8088	186, 188	286
8	12	8, 12	5

Flags : Affectés tous
Indéfinis

Exemples :

POPF codée 9D

PUSH — Sauvegarde d'un mot contenu du registre ou case mémoire, en pile

Le contenu du registre, ou de la case mémoire (16 bits) est rangé en haut de la pile, le contenu du registre SP est décrémenté de 2.

1. Sauvegarde d'un registre

Code machine

0101 0 reg			

Durée

8086	8088	186, 188	286
11	15	10, 14	3

Flags : Affectés aucun
Indéfinis

Exemples :

PUSH AX codée 50
PUSH SI codée 56

2. Sauvegarde d'un registre de segment

Code machine

000 reg 110			

Durée

8086	8088	186, 188	286
10	14	9, 13	3

Flags : Affectés aucun
Indéfinis

Exemples :

PUSH SS codée 16

Note : PUSH CS est autorisé.

3. Sauvegarde du contenu d'une case mémoire (transfert de paramètre)

Code machine

1111 1111	mod 110 r/m		

Durée

	8086	8088	186, 188	286
mémoire	16+AE	24+AE	16, 20	5

Flags : Affectés aucun
Indéfinis

Exemples :

PUSH BETA [BX] codée FF 37 selon code BETA

Note : on peut évidemment coder, en machine, cette dernière instruction pour un registre mais elle nécessite deux octets.

PUSH — Sauvegarde d'une donnée

La donnée (16 ou 8 bits étendus à 16 bits signés) est sauvée en pile, le pointeur de pile SP est décrémenté de 2. Cette instruction permet le passage de paramètres lors de l'appel de procédure (sous-programme).

Code machine

0110 10s0	donnée	donnée si s = 0	

Durée

8086	8088	186, 188	286
—	—	10, 14	3

Flags : Affectés aucun
Indéfinis

Exemples :

```
VAR__1 EQU 2BC8H
PUSH   VAR__1            codée 6A C8 2B
```

PUSHA — Sauvegarde de tous les registres (PUSH ALL)

Les contenus des registres généraux, pointeurs et index, sont sauvés en pile, AX étant stocké le premier et DI le dernier. Le contenu du pointeur de pile (SP) est décrémenté de 16D.

Code machine

0110 0000			

Durée

8086	8088	186, 188	286
–	–	36, 36	17

Flags : Affectés aucun
Indéfinis

Exemples :

```
PUSHA                codée 60
```

Note : PUSHA = est identique à la suite d'instructions

```
PUSH AX
PUSH CX
PUSH DX
PUSH BX
PUSH SP
PUSH BP
PUSH SI
PUSH DI
```

PUSHF — Sauvegarde des flags

Les flags sont stockés en haut de la pile, le contenu du registre SP est décrémenté de 2.

Code machine

1001 1100			

Durée

8086	8088	186, 188	286
10	14	9, 13	3

Flags : Affectés aucun
Indéfinis

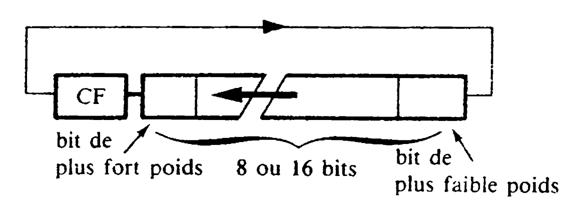
Exemples :

PUSHF codée 9C

RCL — Rotation à gauche avec carry

Cette rotation, à gauche, du contenu d'un registre ou d'une case mémoire, avec CF peut avoir lieu plusieurs fois selon le contenu du registre CL.

La rotation s'effectue selon le schéma ci-dessous.



Il s'agit d'une, ou plusieurs, permutation circulaire, de droite à gauche, de 9 ou 17 bits. Si le nombre de rotations est limité à 1, le flag d'overflow sera significatif : il sera mis à 1 si le bit de plus fort poids du mot résultant est différent de CF, sinon il est mis à zéro (perte ou non de l'information de signe).

Code machine

1101 00vw	mod 010 r/m		

v = 0 nombre de rotation égal à 1
v = 1 nombre de rotation égal au contenu de CL

Durée

	8086	8088	186, 188	286
une rotation registre	2	2	2	2
mémoire	15 + AE	23 + AE	15	7
n rotations registre	8 + 4 * n	8 + 4 * n	5 + n	5 + n
mémoire	20 + AE + 4 * n	28 + AE + 4 × n	17 + n	8 + n

Flags : Affectés CF, OF (significatif pour une rotation)
Indéfinis

Exemples :

RCL AH,1	codée D0 D4
RCL WORD PTR [DI],1	codée D1 15
RCL BX,CL	codée D3 D3
RCL BYTE PTR [SI],CL	codée D2 14

Notes :
— Pour iAPX 186, iAPX 188 et iAPX 286 le nombre de rotations est limité à 31D (5 bits de poids faible du contenu de CL).

— (CL) n'est pas détruit.

RCL — Rotation à gauche avec carry, n fois

Cette instruction, comme la précédente, réalise des permutations circulaires à gauche de 9 ou 17 bits (mot de 8 ou 16 bits, plus le carry). Le nombre de permutations à réaliser est ici fixé par une donnée **qui doit être inférieure à 32D** ou qui sera automatiquement limitée à 31 (5 bits de poids faible).

Code machine

1100 000w	mod 010 r/m	donnée	

w = 0 : mot de 8 bits

Durée

	8086	8088	186, 188	286
n rotations registre	—	—	5 + n	5 + n
mémoire	—	—	17 + n	8 + n

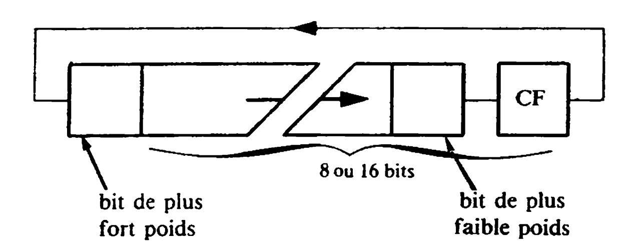
Flags : Affectés CF, OF (significatif pour une rotation)
Indéfinis

Exemples :

RCL BX, 26D	codée C1 D3 1A

RCR — Rotation à droite avec carry

Cette rotation, à droite, du contenu d'un registre ou d'une case mémoire, avec CF peut avoir lieu plusieurs fois selon le contenu du registre CL.
La rotation s'effectue selon le schéma ci-dessous.



Il s'agit d'une, ou plusieurs, permutation circulaire, de gauche à droite, de 9 ou 17 bits.

Si le nombre de rotations est limité à 1, le flag d'overflow est significatif : il sera mis à 1 si le bit de plus fort poids est différent du bit précédent sinon il est mis à zéro (perte ou non de l'information de signe).

Code machine

1101 00vw	mod 011 r/m		

v = 0 : nombre de rotations égal 1

v = 1 : nombre de rotations égal au contenu de CL

Durée

	8086	8088	186, 188	286
une rotation registre	2	2	2	2
mémoire	15 + AE	23 + AE	15	7
n rotation registre	8 + 4*n	8 + 4 × n	5 + n	5 + n
mémoire	20 + AE + 4*n	28 + AE + 4*n	17 + n	8 + n

Flags : Affectés CF, OF (significatif pour une rotation)
Indéfinis

Exemples :

```
RCR AH, 1                 codée D0 DC
RCR WORD PTR [DI],1       codée D1 1D
RCR BX,CL                 codée D3 DB
RCR BYTE PTR [SI],CL      codée D2 1C
```

Notes :
— Pour iAPX 186, iAPX 188 et iAPX 286, le nombre de rotations est limité à 31D (5 bits de poids faible du contenu de CL).

— (CL) n'est pas détruit.

RCR — Rotation à droite avec carry, n fois

Cette instruction, comme la précédente, réalise des permutations circulaires à droite de 9 ou 17 bits (mot de 8 ou 16 bits plus le carry). Le nombre de permutations à réaliser est ici fixé par une donnée qui doit être **inférieure à 32D,** ou qui sera **limitée automatiquement à 31D** (5 bits de poids faible).

Code machine

1100 000w	mod 011 r/m	donnée	

w = 0 : mot de 8 bits

Durée

	8086	8088	186, 188	286
n rotations registre mémoire	—	—	5+n 17+n	5+n 8+n

Flags : Affectés CF, OF (significatif pour une rotation)
Indéfinis

Exemples :

RCR BX, 26D codée C1 DB 1A

REP/REPZ/REPE/REPNE/REPNZ — Répétition d'opérations sur suite(s) de mots

Il s'agit d'un « *préfixe* » qui permet la répétition de certaines opérations concernant une suite de mots — MOVS, CMPS, SCAS, LODS, STOS. L'opération, préfixée, est répétée tant que le contenu de CX n'est pas nul. Dans le cas de CMPS (comparaison de 2 suites pointées par (DI) et (SI)) ou de SCAS (comparaison d'une suite pointée par (DI) au contenu de l'accumulateur), l'opération peut être répétée tant que deux termes sont — REPE — (ne sont pas — REPNE —) égaux. L'arrivée à zéro du contenu de CX interrompt toutefois la répétition si la condition prévue n'a pas été remplie.

Code machine

1111 001z			

Durée

8086	8088	186, 188	286
2	2	2	2

Flags : Affectés voir les instructions concernées
Indéfinis

Exemples :

1) REP MOVS DEST, SOURCE codée F3 A5

2) REPE CMPS DEST, SOURCE codée F3 A7

l'opération sera répétée jusqu'à ce que le contenu de CX soit nul, **à moins que deux termes soient différents.**

3) REPNZ SCAS DEST codée F2 AF

L'opération sera répétée jusqu'à ce que le contenu de CX soit nul, **à moins que deux termes soient égaux.**

Note : en cas d'interruption, le programme est repris, dans ce cas, un octet avant l'instruction interrompue, il faut donc, en cas d'interruptions autorisées, limiter le nombre de préfixes à un.

RET — Retour, fin de sous-programme

Cette instruction termine un sous-programme, mais il faut qu'elle corresponde à l'appel, en ce sens qu'un appel long (intersegment) exige un retour long, et qu'un appel court (intrasegment) exige un retour court.

En effet, on peut, à l'issue d'un retour, charger depuis la pile, soit IP et CS, soit IP seul.

Il est possible de modifier le contenu du pointeur de pile d'une quantité donnée. Ceci permet le passage de paramètres, à l'aide de la pile, avant l'appel d'un sous-programme.

1. Retour court IP seul rechargé

Code machine

1100 0011			

Durée

8086	8088	186, 188	286
16	20	16, 20	11 + m*

*m = nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

RET codée C3

2. Retour court avec modification de SP

Code machine

1100 0010	donnée basse	donnée haute	

Durée

8086	8088	186, 188	286
20	24	18, 22	11+m*

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

RET 4 codée C2 04 00

3. Retour long IP et CS rechargés

Code machine

1100 1011			

Durée

8086	8088	186, 188	286
26	32	22, 30	15+m*

*m : nombre d'octets de l'instruction suivante

Flags : Affectés aucun
Indéfinis

Exemples :

RET codée CB

4. Retour long avec modification de SP

Code machine

1100 1010	donnée basse	donnée haute	

Durée

8086	8088	186, 188	286
25	31	25, 33	15+m*

*m : nombre d'octets de l'instruction suivante

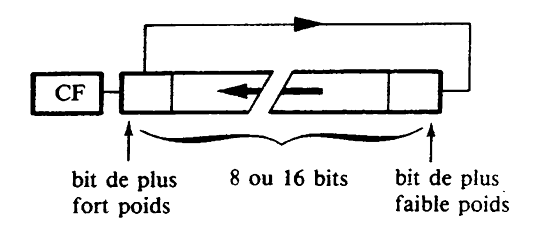
Flags : Affectés aucun
Indéfinis

Exemples :

RET 12 codée CA 0C 00

ROL — Rotation à gauche

Cette rotation à gauche, du contenu d'un registre ou d'une case mémoire peut avoir lieu plusieurs fois selon le contenu de CL. La rotation s'effectue selon le schéma ci-dessous.



Si le nombre de rotations est limité à 1, le flag d'overflow est significatif : il est mis à 1 si le bit de plus fort poids est différent de CF sinon il est mis à 0.

Code machine

1101 00vw	mod 000 r/m		

v = 0 une rotation
v = 1 plusieurs rotations

Durée

	8086	8088	186, 188	286
une rotation registre	2	2	2	2
mémoire	15+AE	23+AE	15	7
n rotations registre	8+4*n	8+4*n	5+n	5+n
mémoire	20+AE+4*n	28+AE+4*n	17+n	8+n

Flags : Affectés CF, OF (significatif pour une rotation)
Indéfinis

Exemples :

ROL AH,1	codée D0 C4
ROL WORD PTR [DI],1	codée D1 05
ROL BX, CL	codée D3 C3
ROL BYTE PTR [SI], CL	codée D2 04

Notes :
— Pour iAPX 186, iAPX 188 et iAPX 286, le nombre de rotations est limité à 31D (5 bits de poids faible du contenu de CL).

— (CL) n'est pas détruit.

— Cette instruction permet de tester un bit sans masque.

ROL — Rotation à gauche, n fois

Cette instruction, comme la précédente, réalise des permutations circulaires, à gauche de 8 ou 16 bits avec copie du bit de plus fort poids dans le carry. Le nombre de permutations à réaliser est ici fixé par une donnée qui doit être **inférieur à 32D** ou qui sera **automatiquement limitée** à 31 (5 bits de poids faible).

Code machine

1100 000w	mod 000 r/m	donnée	

w = 0 : mot de 8 bits

Durée

	8086	8088	186, 188	286
n rotations registre	—	—	5 + n	5 + n
mémoire			17 + n	8 + n

Flags : Affectés CF, OF (significatif pour une rotation)
Indéfinis

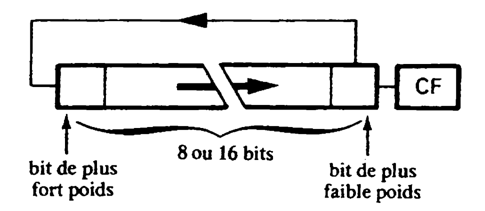
Exemples :

ROL BX, 26D	codée C1 C3 1A

ROR — Rotation à droite

Cette rotation à droite du contenu d'un registre ou d'une case mémoire, peut avoir lieu, plusieurs fois selon le contenu de CL.

La rotation s'effectue selon le schéma ci-dessous :



Si le nombre de rotations est limité à 1, le flag d'overflow est significatif : il est mis à 1 si le bit de plus fort poids est différent du bit précédent (perte d'information du signe) sinon il est mis à 0.

Code machine

1101 00vw	mod 001 r/m			

v = 0 : une rotation
v = 1 : plusieurs rotations

Durée

	8086	8088	186, 188	286
une rotation registre	2	2	2	2
mémoire	15 + AE	23 + AE	15	7
n rotations registre	8 + 4*n	8 + 4*n	5 + n	5 + n
mémoire	20 + AE + 4*n	28 + AE + 4*n	17 + n	8 + n

Flags : Affectés CF, OF (significatif pour une rotation)
Indéfinis

Exemples :

```
ROR AH,1                    codée D0 CC
ROR WORD PTR [DI],1         codée D1 0D
ROR BX, CL                  codée D3 CB
ROR BYTE PTR [SI], CL       codée D2 0C
```

Notes :
Cette instruction permet de tester un bit d'un mot sans avoir recours à un masque.

— Chez iAPX 186, iAPX 188 et iAPX 286, le nombre de rotations est limité à 31 (5 bits de poids faible).

— (CL) n'est pas détruit.

ROR — Rotation à droite, n fois

Cette instruction, comme la précédente, réalise des permutations circulaires, à droite, de 8 ou 16 bits, avec copie du bit de plus faible poids dans le carry. Le nombre de permutations à réaliser est, ici, fixé par une donnée qui doit être **inférieure à 32D** ou qui sera **automatiquement limitée à 31** (5 bits de poids faible).

Code machine

1100 000w	mod 001 r/m	donnée	

w = 0 : mot de 8 bits

Durée

	8086	8088	186, 188	286
n rotations registre	—	—	5+n	5+n
mémoire			17+n	8+n

Flags : Affectés CF, OF (significatif pour une rotation)
Indéfinis

Exemples :

ROR BX, 26D codée C1 CB 1A

SAHF — Mise en place des 5 flags SF, ZF, AF, PF, CF

Les cinq flags sus-nommés prennent les valeurs des bits correspondant du contenu de AH.

SF	ZF	X	AF	X	PF	X	CF

X = indéfini

Code machine

1001 1110			

Durée

8086	8088	186, 188	286
4	4	3	2

Flags : Affectés AF, CF, PF, SF, ZF
Indéfinis

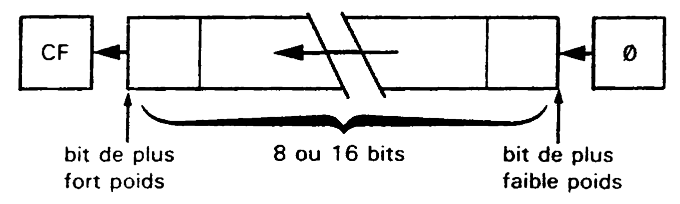
Exemples :

SAHF codée 9E

SAL/SHL — Décalage à gauche, arithmétique ou logique

Ce décalage à gauche du contenu d'un registre ou d'une case mémoire peut avoir lieu plusieurs fois selon le contenu de CL.

Le décalage s'effectue selon le schéma ci-dessous :



ce qui équivaut à une multiplication par 2 à chaque décalage *(attention au signe)*. Si le nombre de décalage est limité à 1, le flag d'overflow est significatif : il est mis à 1 si le bit de plus fort poids est différent de CF (perte de l'information de signe), sinon il est mis à 0.

Code machine

1101 00vw	mod 100 r/m		

v = 0 : un décalage

v = 1 : n décalages

Durée

	8086	8088	186, 188	286
un décalage registre	2	2	2	2
mémoire	15 + AE	23 + AE	15	7
n décalages registre	8 + 4 * n	8 + 4 * n	5 + n	5 + n
mémoire	20 + AE + 4 * n	28 + AE + 4 * n	17 + n	8 + n

Flags : Affectés CF, OF, PF, SF, ZF
Indéfinis AF

Exemples :

```
SAL AH,1                  codée D0 E4
SAL WORD PTR [DI],1       codée D1 25
SAL BX, CL                codée D3 E3
SAL BYTE PTR [SI], CL     codée D2 24
```

Notes :
1) Il est parfois plus intéressant (vitesse) d'utiliser SAL que MUL quand l'un des termes est une puissance de 2.

Par exemple, la multiplication du contenu d'un registre par 8 (2^3) dure :

— avec SAL : MOV CL, 3
SAL reg, CL
soit $4+8+4\times3=19$ périodes
— avec MUL : MOV CL, 8
MUL AL, CL
soit 4 + (70 à 77) = (74 à 81) périodes
qui peuvent devenir (122 à 137) si on écrit MUL AX,CX.

2) Chez iAPX 186, 188 et iAPX 286, le nombre de décalages est limité à 31 (5 bits de poids faible du contenu de CL).

3) (CL) n'est pas détruit.

SAL/SHL — Décalage arithmétique, ou logique, à gauche, n fois

Cette instruction, comme la précédente, réalise des décalages à gauche de 8 ou 16 bits, avec mise à zéro du bit de poids faible. Le nombre de décalages est ici, fixé par une donnée qui doit être **inférieure à 32D** ou qui sera **limitée automatiquement à 31** (5 bits de poids faible).

Code machine

1100 000w	mod 100 r/m	donnée	

w = 0 : mot de 8 bits

Durée

	8086	8088	186, 188	286
n décalages registre	—	—	5 + n	5 + n
mémoire	—	—	17 + n	8 + n

Flags : Affectés CF, OF, PF, SF, ZF
Indéfinis AF

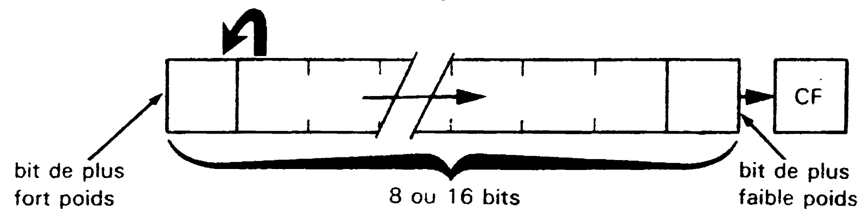
Exemples :

SAL BX, 8 codée C1 E3 08

équivaut à multiplier le contenu de BX par $2^8 = 256$ et ne demande que 5 + 8 périodes alors que IMUL BX, BX, 8 demande 21 périodes (pour iAPX 286).

SAR — Décalage arithmétique à droite

Le décalage à droite du contenu d'un registre ou d'une case mémoire peut avoir lieu plusieurs fois, selon le contenu de CL. Le décalage s'effectue selon le schéma ci-dessous :



Ce qui équivaut à une division par 2, **signée**, par décalage.

Si le nombre de décalage est limité à 1, le flag d'overflow sera mis à zéro, sinon il est indéfini.

Code machine

1101 00vw	mod 111 r/m		

v = 0 : un décalage
v = 1 : plusieurs décalage

Durée

	8086	8088	186, 188	286
un décalage registre	2	2	2	2
mémoire	15 + AE	23 + AE	15	7
n décalages registre	8 + 4 * n	8 + 4 * n	5 + n	5 + n
mémoire	20 + AE + 4 * n	28 + AE + 4 * n	17 + n	8 + n

Flags : Affectés CF, OF, PF, SF, ZF
Indéfinis AF

Exemples :

SAR AH, 1	codée D0 FC
SAR WORD PTR [DI], 1	codée D1 3D
SAR BX, CL	codée D3 FB
SAR BYTE PTR [SI], CL	codée D2 3C

Notes :

— Il est parfois plus intéressant (vitesse) d'utiliser SAR plutôt que IDIV quand le diviseur est une puissance de 2 et que l'on *ne désire que le quotient.*

— Chez iAPX 186, iAPX 188 et iAPX 286, le nombre de décalages est limité à 31 (5 bits de poids faible du contenu de CL).

— (CL) n'est pas détruit.

SAR — Décalage arithmétique, à droite, n fois

Cette instruction, comme la précédente, réalise des décalages à droite de 8 ou 16 bits, avec copie du bit de poids fort. Le nombre de décalages est ici, fixé par une donnée qui doit être **inférieure à 32D** ou qui sera **automatiquement limitée à 31** (5 bits de poids faible).

Code machine

1100 000w	mod 111 r/m	donnée	

w = 0 : mot de 8 bits

Durée

	8086	8088	186, 188	286
n décalages registre	—	—	5 + n	5 + n
mémoire			17 + n	8 + n

Flags : Affectés CF, OF, PF, SF, ZF
Indéfinis AF

Exemples :

SAR BX, 8 codée D1 FB 08

équivaut à diviser le contenu, *signé*, de BX par $2^8 = 256$.

SBB — Soustraction avec retenue *(Borrow)*

Cette instruction permet la soustraction du contenu du deuxième opérande, du contenu du premier. Le résultat prend la place de ce dernier et est diminué de 1, si CF était à 1 avant l'exécution de l'instruction.

L'opération peut avoir lieu entre registres, registre et mémoire, registre ou mémoire et donnée.

1. Opération entre registres, registre et mémoire

Code machine

0001 10dw	mod reg r/m		

d = 1 : le registre désigné par *reg* est la destination du résultat ; s'il y a deux registres, il s'agit du premier nommé.

Durée

	8086	8088	186, 188	286
registre – registre	3	3	3	2
registre – mémoire	9 + AE	13 + AE	10	7
mémoire – registre	16 + AE	24 + AE	10	7

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

SBB AX, BX	codée 1B C3
SBB CH, DL	codée 1A EA
SBB DI, ALPHA [SI]	codée 1B selon ALPHA
SBB BETA [BP], SI	codée 19 selon BETA

2. Opération entre accumulateur (AL ou AX) et donnée (immédiate)

Code machine

0001 110w	donnée	donnée si w = 1	

Durée

8086	8088	186, 188	286
4	4	3 – 4	3

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

SBB AX, 1973 H codée 1D 73 19

3. Opération entre registre ou mémoire et donnée (immédiate)

Code machine

1000 000sw	mod 011 r/m	donnée	donnée si s=0, w=1

si s=0 et w=1 donnée sur 16 bits
s=1 et w=1 donnée sur 8 bits, étendue, signée, à 16 bits

Durée

	8086	8088	186, 188	286
registre	4	4	4	3
mémoire	17+AE	25+AE	16	7

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

SBB AH, 19 H codée 10 DC 19
SBB WORD PTR [DI], 1973 H codée 11 1D 73 19

SCAS — **Analyse d'une suite de mots** ***(Scanning)***

Permet la comparaison du contenu de l'accumulateur (AX ou AL) au contenu de la case mémoire pointée par (DI) par rapport à (ES) par soustraction dont le résultat n'est pas retourné ; seuls les flags sont affectés (SCAS est identique à CMP AX ou AL, [DI]).

Après exécution de l'instruction, le contenu de DI est incrémenté (ou décrémenté) selon la valeur de DF — si DF est nul, il y a incrémentation — d'une unité ou deux, en fonction de la taille des mots comparés.

Cette instruction peut être répétée n fois, si elle est précédée du préfixe REPE ou REPNE.

Code machine

1010 111w			

Durée

	8086	8088	186, 188	286
une fois	15	19	15	7
n fois	9+15*n	9+19*n	5+15*n	5+8*n

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

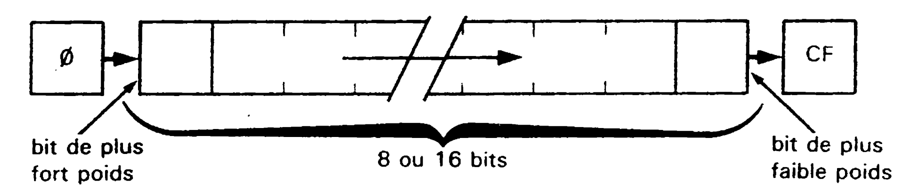
```
1)    CLD              ; met DF à 0, DI incrémenté
      MOV DI, TABLE
      MOV AL, 'M'      ; recherche 'M' = 4DH
      SCASB                    codée AE

2)    STD              ; met DF à 1, DI décrémenté
      MOV DI, TABLE
      MOV AX, 'MD'     ; recherche 'MD' = 4D 44H
      SCASW                    codée AF
```

SHL — voir SAL

SHR — Décalage logique à droite

Ce décalage, à droite, du contenu d'un registre ou d'une case mémoire peut avoir lieu une ou plusieurs fois selon le contenu CL. Il s'effectue ainsi :



Ce qui équivaut à une division par 2 **non signée.** Si le nombre de décalage est limité à 1, le flag d'overflow est significatif : il est mis à 1 si le bit de plus fort poids est différent du précédent (perte de l'information de signe) sinon il est mis à 0.

Code machine

1101 00vw	mod 101 r/m		

v = 0 : un décalage
v = 1 : n décalages

Durée

		8086	8088	186, 188	286
un décalage	registre	2	2	2	2
	mémoire	15+AE	23+AE	15	7
n décalages	registre	8+4*n	8+4*n	5+n	5+n
	mémoire	20+AE+4*n	28+AE+4*n	17+n	8+n

Flags : Affectés CF, OF, PF, SF, ZF
Indéfinis AF

Exemples :

```
SHR AH, 1                   codée D0 EC
SHR WORD PTR [DI], 1        codée D1 2D
SHR BX, CL                  codée D3 EB
SHR BYTE PTR [SI], CL       codée D2 2C
```

Note :
— Chez iAPX 186, iAPX 188 et iAPX 286, le nombre de décalages est limité à 31 (5 bits de poids faible du contenu de CL).

— (CL) n'est pas détruit.

SHR — Décalage logique, à droite, n fois

Cette instruction, comme la précédente, réalise des décalages à droite de 8 ou 16 bits, le bit de poids fort étant mis à zéro. Le nombre de décalages est ici, fixé par une donnée qui doit être **inférieure à 32D**, ou qui sera **automatiquement limitée à 31** (5 bits de poids faible).

Code machine

1100 000w	mod 101 r/m	donnée	

w = 0 : mot de 8 bits

Durée

		8086	8088	186, 188	286
n décalages	reg.	—	—	5+n	5+n
	mém.			17+n	8+n

Flags : Affectés CF, OF, PF, SF, ZF
Indéfinis AF

Exemples :

SHR BX, 8 codée C1 EB 08

équivaut à diviser le contenu de BX, ***non signé,*** par $2^8 = 256$.

STC — Mettre le carry à 1 *(Set)*

Force le carry (CF) à la valeur 1.

Code machine

1111 1001			

Durée

8086	8088	186, 188	286
2	2	2	2

Flags : Affectés CF
Indéfinis

Exemples :

STC codée F9

STD — Mettre à un (SET) l'indicateur de direction (DF)

Le flag DF est mis à 1 provoquant la **décrémentation** du contenu de DI (et de SI) lors d'opérations sur des suites de mots.

Code machine

1111 1101			

Durée

8086	8088	186, 188	286
2	2	2	2

Flags : Affectés DF
Indéfinis

Exemples :

STD codée FD

STI — Mettre à un l'indicateur d'interruption (IF)

L'indicateur d'interruption (IF) est mis à 1 autorisant l'interruption, par événement extérieur, du programme après exécution de l'instruction suivante.

Code machine

1111 1011			

Durée

8086	8088	186, 188	286
2	2	2	2

Flags : Affectés IF
Indéfinis

Exemples :

STI ; autorisation des interruptions

STOS — Stockage d'une suite de mots

Le contenu de l'accumulateur de 8 bits (AL) ou 16 bits (AX) est mis dans la(les) case(s) mémoire(s) pointée(s) par (DI), par rapport à (ES). Après exécution de l'instruction, le contenu de DI est incrémenté de 1 pour AL, de 2 pour AX, si l'indicateur de direction (DF) a été mis à 0 ; sinon il y a décrémentation.
Cette instruction peut être répétée n fois, pour remplir une zone mémoire, si elle est précédée du préfixe REP.

Code machine

1010 101w			

Durée

	8086	8088	186, 188	286
une opération	11	15	10	3
n opérations	9+10*n	9+14*n	6+9*n	4+3*n

Flags : Affectés aucun
Indéfinis

Exemples :

```
1)  STD                 ; DF=1
    MOV DI, ALPHA
    STOSB               ; (ALPHA)=AL, (DI)=(DI)-1

2)  CLD                 ; DF=0
    MOV DI, BETA
    STOSW               ; (BETA)=(AX), (DI)=(DI)+2
```

SUB — Soustraction

Cette instruction soustrait le contenu du deuxième opérande du contenu du premier, le résultat est rangé à la place de ce dernier ; elle peut avoir lieu entre registres, registre et mémoire, registre ou mémoire et donnée.

1. Opération entre registre ou registre et mémoire

Code machine

0010 10dw	mod reg r/m		

d = 1 : le résultat sera mis dans un registre ; entre registres, il s'agira du premier nommé, défini par *'reg'*.

Durée

	8086	8088	186, 188	286
registre – registre	3	3	3	2
registre – mémoire	9+AE	13+AE	10	7
mémoire – registre	16+AE	23+AE	10	7

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

```
SUB AX, BX                  codée 2B C3
SUB CH, DL                  codée 2A EA
SUB DI, ALPHA [SI]
SUB GAMMA [BX] [DI], SI
```

2. Opération immédiate avec l'accumulateur (AL ou AX)

Code machine

0010 110w	donnée	donnée si w = 1	

Durée

8086	8088	186, 188	286
4	4	3 – 4	3

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

SUB AL, 4 codée 2C 04
SUB AX, 660 H codée 2D 60 06

3. Opération immédiate avec registre ou mémoire

Code machine

1000 00sw	mod 101 r/m	donnée	donnée si s = 0, w = 1

la donnée de 8 bits sera étendue à 16 bits, signée, avant opération, si elle est soustraite d'un mot de 16 bits et s et w seront mis à 1.

Durée

	8086	8088	186, 188	286
registre	4	4	4	3
mémoire	17 + AE	25 + AE	16	7

Flags : Affectés AF, CF, OF, PF, SF, ZF
Indéfinis

Exemples :

SUB AH, 45 codée 80 EC 2D

SUB SI, 56 codée 83 EE 38

SUB WORD PTR [SI], 2964 H codée 81 2C 64 29

TEST — Comparaison logique

Cette instruction réalise un ET entre les contenus des deux opérandes mais le résultat n'est pas donné, seuls les flags sont affectés. Le carry et l'overflow sont mis à 0.

L'opération, comme le ET (AND), peut avoir lieu entre registres, registre et mémoire, donnée et registre ou mémoire.

1. Opération entre registres ou registre et mémoire

Code machine

1000 010w	mod reg r/m		

Durée

	8086	8088	186, 188	286
entre registres	3	3	3	2
entre reg. et mém.	9+AE	13+AE	10	6

Flags : Affectés PF, SF, ZF ; CF = OF = 0
Indéfinis AF

Exemples :

```
TEST AX, DX                  codée 85 C2
TEST BH, CL                  codée 84 F9
TEST ALPHA [DI], DX
TEST AL, GAMMA [BP] [SI]
```

2. Opération immédiate avec l'accumulateur (AL ou AX)

Code machine

1010 100w	donnée	donnée si w = 1	

Durée

8086	8088	186, 188	286
4	4	3–4	3

Flags : Affectés PF, SF, ZF ; CF = OF = 0
Indéfinis AF

Exemples :

TEST AL, 6 codée 08 06
TEST AX, 1984 H codée A9 94 18

3. Opération immédiate avec registre ou mémoire

Code machine

1111 011w	mod 000 r/m	donnée	donnée si w = 1

Durée

	8086	8088	186, 188	286
registre	5	5	4	3
mémoire	11 + AE	19 + AE	10	6

Flags : Affectés PF, SF, ZF, CF = OF = 0
Indéfinis AF

Exemples :

TEST AH, OFH codée F6 C4 OF
TEST WORD PTR [BP] [DI], 6ACEH · codée F7 03 CE 6A

WAIT — Attendre

Cette instruction met le microprocesseur en attente jusqu'à réception d'un signal sur la patte $\overline{\text{TEST}}$ (passant de 5 V à 0 V). Cet état peut être interrompu par une interruption externe, le retour a lieu *dans ce cas,* à l'instruction WAIT elle-même. Elle sert à synchroniser le processeur sur des circuits externes. Elle fait partie de certaines instructions du 8087.

Code machine

1001 1011			

Durée

8086	8088	186, 188	286
3 + 5n	3 + 5n	6	3

Flags : Affectés aucun
Indéfinis

Exemples :

WAIT codée 9B

XCHG — Echange

Provoque l'échange des contenus des deux opérandes. L'opération peut avoir lieu entre registres, registre et mémoire.

1. Opération avec l'accumulateur AX et un registre

Code machine

100 10reg			

Durée

8086	8088	186, 188	286
3	3	3	3

Flags : Affectés aucun
Indéfinis

Exemples :

XCHG AX, BX codée 93

2. Opération entre registres ou registre et mémoire

Code machine

1000 011w	mod reg r/m		

Durée

	8086	8088	186, 188	286
entre registres	4	4	4	3
registre et mémoire	17+AE	25+AE	17	5

Flags : Affectés aucun
Indéfinis

Exemples :

```
XCHG AH, AL          codée 86 C4
XCHG BL, AL          codée 86 D8
XCHG DH, ALPHA
```

XLAT — Traduire

Cette instruction met dans l'accumulateur AL, le contenu de la case mémoire d'adresse définie par la somme des contenus de BX et de AL par rapport à (DS).

Code machine

1101 0111			

Durée

8086	8088	186, 188	286
11	11	11,15	5

Flags : Affectés aucun
Indéfinis

Exemples :

```
LEA BX, TAB_ASCII
XLAT TAB_ASCII          codée D7
```

— Si BX pointe le début d'une table ASCII, nous aurons dans AL le code ASCII du précédent contenu.

— *Ne pas* écrire TAB_ASCII comme un label.

XOR — Ou exclusif

Permet de réaliser un « ou exclusif » entre les contenus des deux opérandes, le résultat est rangé à la place du premier nommé. Les flags CF et OF sont mis à zéro. L'opération peut avoir lieu entre registres, registre et mémoire, donnée et registre ou mémoire.

1. Opération entre registres ou registre et mémoire

Code machine

0011 00dw	mod reg r/m		

d = 1 : le résultat sera mis dans le registre, le premier nommé, s'il s'agit de deux registres, sera défini par *reg*.

Durée

	8086	8088	186, 188	286
registre⊕registre	3	3	3	2
registre⊕mémoire	9+AE	13+AE	10	7
mémoire⊕registre	16+AE	24+AE	10	7

Flags : Affectés PF, SF, ZF ; CF = OF = 0
Indéfinis AF

Exemples :

XOR AH, BL codée 32 E3

XOR CX, DI codée 33 CF

XOR AX, ALPHA [DI]

XOR GAMMA [SI], BX

2. Opération immédiate avec l'accumulateur AL ou AX

Code machine

0011 010w	donnée	donnée si w = 1	

Durée

8086	8088	186, 188	286
4	4	3 – 4	3

Flags : Affectés PF, SF, ZF ; CF = OF = 0
Indéfinis AF

Exemples :

XOR AL, OF6H	codée 34 F6
XOR AX, 400H	codée 35 00 04

3. Opération immédiate avec registre ou mémoire

Code machine

1000 000w	mod 110 r/m	donnée	donnée si w = 1

Durée

	8086	8088	186, 188	286
registre	4	4	4	3
mémoire	17+AE	25+AE	10	7

Flags : Affectés PF, SF, ZF ; CF = OF = 0
Indéfinis AF

Exemples :

XOR AH, OF6H	codée 80 F4 F6
XOR BYTE PTR [DI], 33	codée 80 35 21

INSTRUCTIONS DE CONTROLE EN MODE PROTEGE (iAPX 286 seulement)

A la mise sous tension ou après un « Reset » manuel, iAPX 286 travaille en mode réel *(Real Address Mode)* dans un espace de 1 M octets avec recouvrement éventuel des segments (mode de travail du 8086). En **mode protégé, les segments sont disjoints**, l'espace physiquement adressable est de 16 M octets (24 bits) et l'espace virtuel de 1 G octets (1000 M soit 30 bits).
Pour travailler ainsi, iAPX 286 nécessite :

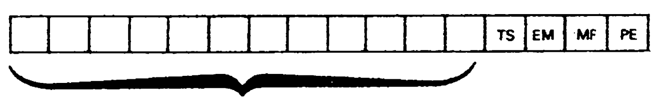
— 2 flags supplémentaires :

• NT *(Nested Taste)* flag de tâche emboîtée,

• I/OPL de 2 bits fixant le niveau de privilège *(Privilege Level)* des entrées/sorties *(In/Out)*.
Le registre des flags de iAPX 286 est alors :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NT	I/O	PL	OF	DF	IF	TF	SF	ZF		AF		PF		CF

— Un mot d'état machine (MSW : *Machine Status Word) :*



Réservé

avec :

• PE *(Protection Enable)* protection activée si le bit est à 1, il ne peut être mis à 0 que par un reset manuel.

• MP *(Monitor Processor Extension)* qui permet la mise au point en autorisant les interruptions de contrôle (existence ou non d'une extension).

- EM *(Emulate Processor Extension)* qui permet l'émulation de l'extension par logiciel.
- TS *(Task Switch)* assure une protection contre les erreurs de l'extension.

Nous avons pour ces 3 flags les possibilités suivantes :

TS	MP	EM	
0	0	0	— après un Reset : mode réel
0	0	1	— pas d'extension ESC provoque une interruption du type 7
1	0	1	
0	1	0	— l'extension existe
1	1	0	— l'extension existe et ESC ou WAIT génère une interruption de type 7

En mode protégé, iAPX 286 travaille en multitâches, par exemple : entrée clavier, affichage sur écran, impression... A chaque tâche correspond un niveau de privilège (PL), le niveau 0 étant celui de *l'Operating System* (OS) qui est ainsi isolé des programmes d'application isolés les uns des autres. On trouve, en cours de traitement d'une tâche :

— le CPL **(Current Privilege Level)** qui est le niveau de privilège de la tâche,

— le RPL **(Request Privilege Level)** qui est le niveau demandé,

— le DPL **(Descriptor Privilege Level)** qui est le niveau de privilège d'une zone mémoire.

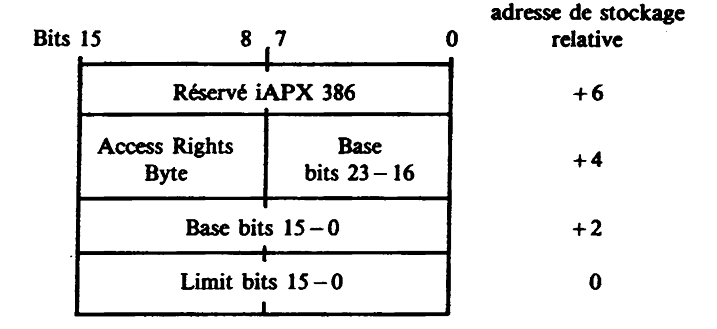
Les opérations ne peuvent avoir lieu que si :
CPL est inférieur ou égal à DPL
et RPL inférieur ou égal à DPL.

Dans un tel système une zone mémoire physique, définie par son segment CS, DS, SS ou ES, ne peut être atteinte que si elle est d'un niveau de privilège inférieur à celui de la tâche en cours. Cette zone a une adresse sur 24 bits, une étendue de 64 K au maximum. Pour la définir, il faut donc :

1) 1 mot pour l'étendue appelée *LIMIT*

2) 3 octets pour l'adresse de base (24 bits) auxquels on ajoute un octet définissant entre autre, le niveau de privilège et si la zone mémoire peut être lue ou écrite. Cet octet définit les règles d'accès à la zone considérée *(ACCESS RIGHTS BYTE)* en particulier, deux bits définissent DPL.

L'ensemble de ces 3 mots complété par un mot réservé pour iAPX 386 (microprocesseur 32 bits) constitue la table de description du segment *(Segment Descriptor Table).*



Le champ de 1 G octets est découpé en 16 K tranches de 64 K octets. Les registres de segments sont appelés *selector* et deviennent des « registres » de 64 bits, dont 16 sont dits *visibles,* les deux bits de poids faible constituant RPL, et 48 cachés *(hidden)* contenant les informations décrites ci-dessus et appelé *descriptor.*

Pour plus d'informations, reportez-vous au *Programmer's Reference Manual* de l'iAPX 286, puisque nous ne détaillerons, ici, que les instructions vous permettant de travailler en mode protégé, certaines génèrent une interruption de type 6 en mode réel.

Quelques instructions, déjà vues, ne peuvent être exécutées que lors d'une tâche de niveau 0 ou dont le niveau (CPL) est au plus égal à IOPL qui ne peut être modifié que par une tâche de niveau 0. Il s'agit de :

CLI (IF = 0)

STI (IF = 1)

HLT seulement au niveau 0

LOCK si CPL > IOPL

POPF (Pop Flags) qui peut modifier IOPL est donc réservé au niveau 0

IRET qui restitue les flags

IN, INS
OUT, OUTS } si CPL > IOPL

qui provoquent une interruption de type 13 si elles ne peuvent être exécutées.

ARPL — Ajustement au rang demandé (RPL = *Request Privilege Level)*

Le rang est défini par les 2 bits de poids faible du sélecteur (rang de 0 à 3).

Le premier opérande, mémoire ou registre, contient la valeur du sélecteur, le deuxième est le contenu d'un registre. Si RPL_1 (2 bits de poids faible) du premier opérande est plus petit que celui du deuxième opérande (RPL_2) alors ZF est mis à 1 et RPL_1 est porté à la valeur de RPL_2 sinon ZF est mis à 0.

Code machine

0110 0011	mod reg r/m		

Durée : 10, mémoire : 11

Flags : Affectés ZF
Indéfinis aucun

Exemples :

ARPL CX, AX

Note : ARPL n'est pas reconnue en mode réel et provoque une interruption de type 6.

CLTS — Mettre à zéro le flag de tâche

Cette instruction met le flag de tâche (TS : *Task Switch)* du mot d'état machine (MSW) à zéro, ***autorisant le mode protégé.***

Si le flag MP est à 1, à chaque instruction Wait ou Escape, TS est mis à 1 ce qui peut provoquer une interruption de type 7 à la prochaine instruction Wait ou Escape.
Cette instruction ne peut être exécutée qu'au niveau 0, elle n'apparaît pas dans les programmes d'applications.

Code machine

0000 1111	0000 0110		

Durée : 2

Flags : Affectés TF = 0
Indéfinis aucun

Exemples :

CLTS codée 0F 06

LAR — Chargement de l'octet des règles d'accès *(Access Rights Byte)*

Cette instruction suppose que le deuxième opérande, registre ou mémoire, contient un sélecteur. Si le *descriptor* associé est visible, c'est-à-dire si son niveau (DPL) est supérieur au niveau de la tâche (CPL) et à celui du sélecteur (RPL) [DPL ⩾CPL et DPL ⩾RPL] alors les règles d'accès du *descriptor* sont chargées dans l'octet de poids fort du premier opérande (registre) dont l'octet de poids faible est mis à zéro. Le flag ZF est mis à 1. Si la zone mémoire est inaccessible, le flag ZF est mis à 0.

Code machine

0000 1111	0000 0010	mod reg r/m	

Durée : 14, mémoire : 16

Flags : Affectés ZF
Indéfinis aucun

Exemples :

LAR AX, CX codée 0F 02 C1

Note : cette instruction génère une interruption de type 6 en mode réel.

LGDT — Chargement de la Table de Description Globale (GDT)

La table de description globale est une table commune à plusieurs tâches qui utilisent des tables locales.

Cette instruction permet le chargement de la table de description globale (GDT) à partir des 6 octets pointés par l'adresse effective fournie par l'opérande. LIMIT est chargé avec le premier mot, les 3 octets suivant forment la BASE, le dernier octet est ignoré (règles d'accès).

Code machine

0000 1111	0000 0001	mod 010 r/m	
		mod # 11	

Durée : 11

Flags : Affectés aucun
Indéfinis

Exemples :

LGDT [BX] codée 0F 01 17

Note :
— Cette instruction est autorisée en mode réel pour les initialisations avant la mise en place du mode protégé.

— Elle n'est pas utilisée dans les programmes d'applications.

LIDT — Chargement de la Table de Description des Interruptions (IDT)

Cette instruction assure le chargement de la table de description des interruptions (IDT) à partir de 6 octets pointés par l'adresse effective fournie par l'opérande. LIMIT est chargé avec le premier mot, les 3 octets suivants forment la BASE, le dernier octet est ignoré (règles d'accès).

Code machine

0000 1111	0000 0001	mod 011 r/m	

mod # 11

Durée : 12

Flags : Affectés aucun
Indéfinis

Exemples :

LIDT [BX] codée 0F 01 1F

Note :
— Cette instruction est autorisée en mode réel.

— Elle n'est pas utilisée dans les programmes d'applications.

LLDT — Chargement du registre de la Table de Description Locale (LDT)

Cette instruction permet de charger la table de description locale (LDT) propre à une tâche.

L'opérande doit contenir un sélecteur pointant la table globale (GDT). L'entrée de la table globale doit être la table locale, s'il en est ainsi, la table locale est chargée depuis cette entrée.

Code machine

0000 1111	0000 0000	mod 010 r/m	

Durée : 17, mémoire : 19

Flags : Affectés aucun
Indéfinis

Exemples :

LLDT BX codée 0F 00 D3

Note :
— Cette instruction n'est pas reconnue en mode réel.

— Elle n'apparaît pas dans les programmes d'applications.

LMSW — Chargement du mot d'état machine (MSW)

Cette instruction permet le chargement de mots d'état machine (MSW) à partir du contenu de l'opérande. Elle peut être utilisée pour passer en mode protégé, dans ce cas elle **doit** être suivie d'un saut-intrasegment pour vider la file d'attente *(queue)*. On ne peut l'utiliser pour revenir au mode réel.

Code machine

0000 1111	0000 0001	mod 110 r/m	

Durée : 3, mémoire : 6

Flags : Affectés aucun
Indéfinis

Exemples :

LMSW BX codée 0F 01 F3

Note :
— Cette instruction n'apparaît pas dans les programmes d'applications.

— En mode réel, elle génère une interruption 13 si l'adresse effective (AE) vaut 0FFFFH.

LSL — Chargement de la dimension d'un segment (LIMIT)

Si le *descriptor* défini par le sélecteur fourni par le deuxième opérande (registre ou mémoire) est « visible », c'est-à-dire si DPL est supérieur ou égal à CPL et RPL également supérieur ou égal à CPL, le mot définissant la dimension *(LIMIT)* du *descriptor* sera chargé dans le premier opérande qui doit être un registre. Le flag ZF sera mis à 1. Sinon le flag ZF sera mis à 0.

L'interprétation de la dimension dépend du type de segment. Seules les dimensions des segments, segment de tâche (TSS ou *Task State Segments)* et des tables de description **locales** sont chargées.

Code machine

0000 1111	0000 0011	mod reg r/m	

Durée : 14, mémoire : 16

Flags : Affectés ZF
Indéfinis

Exemples :

LSL AX, CX codée 0F 03 C3

Note :
Cette instruction génère une interruption de type 6 en mode réel.

LTR — Chargement du registre de tâche

Le registre de tâche est chargé depuis l'opérande qui peut être un registre ou une case mémoire. Le segment d'état de tâche (TSS) est *marqué* en *activité*. Un basculement du flag de tâche (TS) n'a pas lieu.

Cette instruction n'apparaît pas dans les programmes d'applications.

Code machine

0000 1111	0000 0000	mod 011 r/m	

Durée : 17, mémoire : 19

Flags : Affectés aucun
Indéfinis

Exemples :

LTR [BX] codée 0F 00 1F

Note : cette instruction génère une interruption de type 6 en mode réel.

SGDT — Stockage de la Table de Description Globale (GDT)

Les contenus des registres de la table de description globale sont copiés dans les six octets définis en mémoire par l'opérande. LIMIT est stocké en premier, puis les trois octets de BASE, le sixième est *indéfini* (règles d'accès).

Cette instruction n'apparaît pas dans les programmes d'applications.

Code machine

0000 1111	0000 0001	mod 000 r/m	

mod≠11

Durée : 11

Flags : Affectés aucun
Indéfinis

Exemples :

SGDT [DI] codée 0F 01 05

Note : cette instruction est reconnue en mode réel.

SIDT — Stockage de la Table de Description des Interruptions (IDT)

Les contenus des registres de la table de description des interruptions sont copiés dans les six octets définis en mémoire par l'opérande.

LIMIT est stocké en premier, puis les 3 octets de BASE, le sixième est *indéfini.*

Cette instruction n'apparaît pas dans les programmes d'applications.

Code machine

0000 1111	0000 0001	mod 001 r/m	

mod≠11

Durée : 12

Flags : Affectés aucun
Indéfinis

Exemples :

SIDT [DI] codée 0F 01 0D

Note : cette instruction est autorisée en mode réel.

SLDT — Stockage du registre de la Table de Description Locale (LDT)

Le registre de la table de description locale est stocké dans les 2 octets définis par l'opérande, registre ou mémoire.

Ce registre est un sélecteur qui pointe à l'intérieur de la table de description globale.

Cette instruction n'est pas utilisée dans les programmes d'applications.

Code machine

0000 1111	0000 0000	mod 000 r/m	

Durée : 2, mémoire : 3

Flags : Affectés aucun
Indéfinis

Exemples :

SLDT [DI] codée 0F 00 05

Note : cette instruction génère une interruption de type 6 en mode réel.

SMSW — Stockage du mot d'état machine

Le mot d'état machine MSW est stocké dans le registre ou les cases mémoires (16 bits) défini par l'opérande.

Code machine

0000 1111	0000 0001	mod 100 r/m	

Durée : 2, mémoire : 3

Flags : Affectés aucun
Indéfinis

Exemples :

SMSW codée 0F 01 02

Note : cette instruction génère, en mode réel, une interruption du type 13 si l'adresse effective est 0FFFFH.

STR — Stockage du registre de tâche

Le contenu du registre de tâche est copié dans le registre ou les deux octets définis par l'opérande.

Code machine

0000 1111	0000 0000	mod 001 r/m	

Durée : 2, mémoire : 3

Flags : Affectés aucun
Indéfinis

Exemples :

STR [BP] [DI] codée 0F 00 0B

Note : cette instruction génère en mode réel une interruption de type 6.

VERR — Vérifier si la lecture est possible

L'opérande doit contenir un sélecteur, l'instruction vérifie si le segment défini est accessible à la tâche et s'il est possible (autorisé) de lire les octets qu'il contient. Si l'opération est possible, le flag ZF est mis à 1. Si le segment n'est pas accessible, le flag ZF est mis à 0.

Code machine

0000 1111	0000 0000	mod 100 r/m	

Durée : 14, mémoire : 16

Flags : Affectés ZF
Indéfinis

Exemples :

VERR [SI] codée 0F 00 24

Note : cette instruction génère une interruption de type 6, en mode réel.

VERW — Vérifier si l'écriture est possible

L'opérande doit contenir un sélecteur. L'instruction vérifie si le segment défini est accessible à la tâche et s'il est possible (autorisé) d'y écrire des données. Si l'opération est possible, le flag ZF est mis à 1. Si le segment n'est pas accessible, le flag ZF est mis à 0.

Code machine

0000 1111	0000 0000	mod 101 r/m	

Durée : 14, mémoire : 16

Flags : Affectés ZF
Indéfinis

Exemples :

VERW [DI] codée 0F 00 2D

Note : cette instruction génère, en mode réel, une interruption de type 6.

ASM 86

En raison des diverses possibilités offertes par les instructions que nous venons de voir et par les différents modes d'adressage qui ont illustré les exemples, il est nécessaire de préciser au programme assembleur, le type d'instruction que l'on veut générer, par exemple :

— chargement en mémoire d'un octet ou d'un mot (16 bits),
— chargement d'un registre à l'aide d'une donnée ou à partir d'une case mémoire,
— saut, appel, indirect long ou court.

Il est donc nécessaire de respecter certaines règles pour éviter une perte de temps non négligeable en raison d'erreurs de syntaxe.

On peut commencer l'écriture d'un programme en lui donnant un nom, précédé de **NAME** et éventuellement un titre qui sera imprimé à chaque page selon le modèle suivant :

$TITLE *('titre')*

Le symbole **$** signale qu'il s'agit d'une option. Nous conseillons d'inclure dans le titre le numéro de la version du programme et la date.

Le programme doit se terminer par END. Cette consigne — *directive* — indique, évidemment au programme assembleur que la liste de nos instructions est terminée et qu'il peut entreprendre une deuxième « *passe* » afin de déterminer, en particulier, les adresses qu'il n'a pu définir à la première lecture.

Dans l'écriture du programme, nous travaillons en *symbolique* c'est-à-dire que nous attribuons des noms aux cases mémoires, aux lignes d'instructions, ce qui rend le programme plus lisible et plus clair.

En assembleur, un programme s'écrit en utilisant des colonnes — *fields* — aux fonctions bien définies : identificateurs ou étiquettes, codes opérations, opérandes et commentaires précédés de ";" qui peuvent éventuellement commencer en première colonne.

I — GENERALITES

I-1. Identificateurs

Les identificateurs permettent de repérer une donnée, une ligne dans un programme (*label* ou étiquette), une case mémoire, un sous-programme ou *procédure*... Ils peuvent être composés de 31 caractères ou plus, les caractères supplémentaires n'étant pas décodés.

Les caractères peuvent être des lettres, majuscules ou minuscules, des chiffres, et quelques caractères spéciaux : @, ?, _. **Les identificateurs ne doivent pas commencer par un chiffre,** ils se terminent par ":" (label), un espace, une tabulation ou un retour chariot. Nous pouvons écrire, par exemple :

AFF_CODES_HEX ou
aff_codes_hex
@1

Le dernier exemple est très pratique pour repérer les sauts courts, conditionnés, à l'intérieur d'un programme. (Evitez $ + x en ASM86 car vous aurez quelques difficultés à évaluer x, $ étant ici le contenu de IP). N'hésitez donc pas à écrire des *identificateurs longs mais clairs,* le programme peut s'en trouver autodocumenté.

I-2. Constantes numériques

Comme pour 8085, on peut donner les constantes en :

— binaire : 01101010B, très utile pour écrire les masques,
— décimal : 65 ou 65D,
— hexadécimal : 72H, 0F7H ; ici, ne pas oublier de mettre 0 devant une lettre ;
— octal (base 8) : 72O ou 72Q.

Toute constante peut être précédée du signe −, elle doit pouvoir être écrite avec 16 bits donc être inférieure ou égale, en valeur absolue, à : 65535D, 177777Q, 0FFFFH ; donc, méfiez-vous des constantes définies à partir d'autres à l'aide d'opérations arithmétiques.

Une constante peut être définie par un nom en utilisant la directive EQU, par exemple :

```
DUREE        EQU 12
PORT_IN      EQU 02
PORT_OUT     EQU 04
```

I-3. Chaînes de caractères

Les chaînes de caractères sont définies entre apostrophes, chaque caractère sera traduit en code ASCII (un octet par caractère).
Exemple :

'A' est équivalent à 41H
'Ag' à 4167H
'[' à 5BH

I-4. Variables

Les variables peuvent être de différents **TYPE** selon les directives suivantes :

— **DB** *Define byte,* la variable est un octet soit de TYPE 1,
— **DW** *Define word,* la variable est un mot (2 octets) soit de TYPE 2,
— **DD** *Double word,* la variable est constituée de 2 mots soit de TYPE 4,
— **DQ** *Quadruple word,* la variable est constituée de 4 mots soit de TYPE 8,
— **DT** *Ten bytes,* la variable est constituée de 10 octets soit de TYPE 10.

Les deux derniers cas sont utilisés pour définir les variables destinées au 8087 qui peut traiter des grandeurs de 80 bits.

Les variables ont une longueur — **LENGTH** — et une dimension **SIZE.**

La longueur correspond au nombre de termes de la variable (nombre d'octets, de mots, de double mots...).

La dimension est le nombre d'octets, nous avons donc la relation SIZE = LENGTH*TYPE. Par exemple :

```
CHIFFRES    DB    0,1,2,3,4,5,6,7,8,9
```

a une longueur de 10D et une dimension de 10 D

```
ANNEES      DW    1980, 1981, 1982, 1983
```

a une longueur de 4 et une dimension de 8.

Attention, une chaîne de caractères ne peut être définie par DW que si elle ne comportent qu'un terme (LENGTH = 1) dans le cas contraire, il faut utiliser DB, par exemple :

```
MESSAGE    DB    'ASM 86'
```

Nous verrons plus loin que l'on peut utiliser des **STRUCTURES,** multi-octets et des **RECORDS** travaillant au niveau du bit.

Remarques : éviter en ASM86 de donner à la chaîne une caractéristique de label en faisant suivre l'identificateur de ":"

I-5. Opérateurs arithmétiques et logiques

Nous avons parfois besoin de définir une grandeur par rapport à d'autres, il est donc possible d'utiliser des définitions faisant appel à des opérations arithmétiques et logiques qui sont :

— les quatre opérations : +, −, *, /
— les quatre opérateurs logiques : AND, OR, XOR, NOT les **opérandes ne pouvant être que des constantes chiffrées**
— MOD, *Modulo,* qui est le reste d'une division :

19/7 = 2
19MOD7 = 5

— les opérateurs de décalages SHL et SHR (décalage à gauche et à droite), le nombre de décalages d'un bit pouvant être fixé par une constante ou symboliquement.
— les opérateurs HIGH et LOW qui permettent de travailler avec l'octet de poids fort (HIGH) ou faible (LOW) d'une donnée de 16 bits, cette donnée pouvant être une adresse définie par un label.

Remarques :
— Eviter en ASM86 de donner à la chaîne une caractéristique de label en faisant suivre l'identificateur de ":".
— Après DB, on peut écrire jusqu'à 255 caractères. Pour que LENGTH puisse être utilisé, on emploie & :

```
MESSAGE   DB   'JE SUIS PRET A VOUS
&              ECOUTER'
```

II — SEGMENTATION

Les instructions sont contenues dans un segment défini par CS, les opérations liées à la pile ont lieu dans un segment défini par SS et les données sont généralement contenues dans le segment défini par DS mais on peut les ranger dans d'autres segments. Il est donc nécessaire de définir les segments de travail.
La définition d'un segment comprend :

— son nom, un identificateur suivi de SEGMENT,
— son adresse ou plus exactement le type d'adressage physique que l'on désire,
— sa nature qui permettra, éventuellement, des regroupements lors de l'utilisation de LINK 86 et de LOC86 (rassemblement de différents programmes et définition des adresses physiques),
— sa classe qui permettra éventuellement le regroupement de segments définis par le même registre de segment lors de l'utilisation de LOC86,
— sa borne supérieure donnée par ENDS précédée du nom du segment.

II-1. Adressage *(align-type)*

Si rien n'est précisé, le segment commencera à une adresse divisible par 16, c'est-à-dire dont le dernier chiffre sera 0 en hexadécimal. Il en sera de même si l'on écrit **PARA** (paragraphe).
L'adresse de base sera quelconque si l'on a écrit **BYTE**, paire s'il s'agit de **WORD**, divisible par 256 — octet de poids faible égal à 00H — pour **PAGE.**

Si l'indication est INPAGE, l'étendue du segment sera au maximum d'une page (256 octets) commençant aux adresses divisibles par 256.

II-2. Nature du segment *(combine-type)*

Si aucune indication n'est fournie le segment ne pourra être associé à aucun autre lors du « LINK ».

Si le segment est déclaré **PUBLIC**, il sera associé à d'autres segments de ***même nom*** également déclarés PUBLIC formant un segment d'étendue égale à la **somme** des étendues.

Si le segment est déclaré **COMMON**, il pourra également être associé à d'autres segments de même caractéristiques, mais l'étendue résultante est **celle du plus grand** segment rencontré.

Si le segment est déclaré **STACK** (pile), tous les segments de mêmes caractéristiques seront combinés de sorte à constituer un segment d'étendue égale à la somme des étendues, mais en fixant l'adresse haute commune à tous les segments ainsi associés. (Ils ont tous le même haut de pile : T.O.S. — *Top Of Stack.)*

Si le segment est déclaré **MEMORY** tous les segments de mêmes caractéristiques seront regroupés mais l'étendue du segment résultant sera égale à celle du premier segment MEMORY rencontré.
Il est possible de fixer l'adresse de base au segment à l'aide de **AT** suivi d'une donnée. Cette possibilité est associée, en général, à ORG pour définir des adresses physiques fixes par exemples celles des interruptions.

II-3. Classe *(classname)*

La classe autorise la juxtaposition des segments de même classe lors de l'utilisation de LOC 86. La classe s'écrit entre apostrophes.

II-4. Multiples définitions

Vous avez le droit d'ouvrir et de fermer (ENDS) un segment plusieurs fois, les différents morceaux ne constitueront qu'un segment à l'assemblage. Il est inutile de répéter à chaque ouverture les attributs du segment mais vous ne pouvez en changer.

II-5. Exemple

```
DATA SEGMENT BYTE PUBLIC 'ROM'
 :
DATA ENDS
```

Le nom du segment est DATA, son mode d'adressage BYTE, sa nature PUBLIC, sa classe 'ROM'.
Ce segment sera associé aux autres segments DATA. Ici PUBLIC est superflu puisque la classe remplit le même rôle, mais il ne pourra être associé à :

```
DATA SEGMENT WORD 'ROM'
 :
DATA ENDS
```

Le changement d'adressage génère un message d'erreur. La classe 'ROM' permet de retrouver les données dans le programme objet (codes hexadécimaux) afin de les stocker en EPROM.

II-6. Segmentation et variables

A l'intérieur d'un segment, les variables sont caractérisées par :
— le segment dont elles dépendent,
— leur adresse à l'intérieur du segment ou **OFFSET**
s'ajoutant aux attributs déjà vus — **TYPE, LENGTH, SIZE.**
Ceci a une très grande importance dans l'écriture des instructions.
Par exemple :

```
TABLES      SEGMENT
TAB_ASCII   DB          '0123456789ABCDEF'
TAB_REG     DB          'AXBXCXDXSIDIBPSP'
TABLES      ENDS
```

Nous avons évidemment :

```
TYPE TAB_ASCII = 1
TYPE TAB_REG = 1
LENGTH TAB_ASCII = 16D
LENGTH TAB_REG = 16D
SEG TAB_ASCII = adresse physique du segment TABLES
OFFSET TAB_ASCII = 0
OFFSET TAB_REG = 10H
```

Ceci est très important pour l'écriture de programme, en effet : voir exemple 1 (ci-contre).

— MOV BX, TAB_ASCII signifie que l'on met dans BX le contenu de la case mémoire d'adresse TAB_ASCII par rapport au segment TABLES alors que TAB_ASCII est définie comme étant un **ensemble d'octets** nous avons génération d'une erreur, ce qui ne se produit pas avec MOV BL, TAB_ASCII ; codée 8AIE 00 00 ;

— MOV BX, OFFSET TAB_REG (codée BB 10 00) met dans BX l'adresse effective de TAB_REG, c'est-à-dire 0010H ;

— MOV AX, SEG TAB_ASCII, codée B8----, met dans AX le contenu du segment CS, DS, ES ou SS qui a été défini comme contenant TABLES, ici---- avec R, comme Relogeable, car DS n'a pas été défini physiquement. (**ASSUME** est précisé en II-7.)

Par contre, si l'on écrit (exemple 2, voir page 150) :

```
TABLES      SEGMENT     AT 70H
            ORG         8FH
TAB_ASCII   DB          '0123456789ABCDEF'
TAB_REG     DB          'AXBXCXDXSIDIBPSP'
TABLES      ENDS
```

nous définissons, avec AT, l'adresse physique du segment TABLES (70H) et l'offset de TAB_ASCII dans ce segment grâce à **ORG** et nous avons :

— MOV AX, WORD PTR TAB_ASCII, codée A1 8F 00, ici sans erreur puisque nous avons précisé grâce à la directive *PTR* (voir plus loin son emploi) que nous désirons met-

```
MCS-86 MACRO ASSEMBLER    ESSAI

ISIS-II MCS-86 MACRO ASSEMBLER V2.1 ASSEMBLY OF MODULE ESSAI
OBJECT MODULE PLACED IN :F1:ESSAI.OBJ
ASSEMBLER INVOKED BY:  ASM86 :F1:ESSAI.SRC


LOC  OBJ                    LINE     SOURCE

----                           1     TABLES  SEGMENT
                               2
0000 30313233343536            3     TAB_ASCII  DB   '0123456789ABCDEF'
     37383941424344
     4546
                               4
0010 41584258435844            5     TAB_REG    DB   'AXBXCXDXSIDIBPSP'
     58534944494250
     5350
                               6
----                           7     TABLES  ENDS
                               8
----                           9     CODESEG SEGMENT
                              10
                              11     ASSUME  CS:CODESEG,DS:TABLES
                              12
0000 9090909090               13     MOV     BX,TAB_ASCII
*** ERROR #2, LINE #13, OPERANDS DO NOT MATCH THIS INSTRUCTION
                              14
0005 8A1E0000                 15     MOV     BL,TAB_ASCII
                              16
0009 BB1000                   17     MOV     BX,OFFSET TAB_REG
                              18
000C B8----          R        19     MOV     AX,SEG TAB_ASCII
                              20
----                          21     CODESEG ENDS
                              22             END

ASSEMBLY COMPLETE, 1 ERROR FOUND
```

Exemple 1

tre dans AX le contenu (16 bits = WORD) de la case mémoire d'adresse TAB_ASCII, c'est-à-dire 008F dans le segment DS ;

— MOV BX, OFFSET TAB_REG, est codée BB 9F 00 ;
— MOV AX, SEG TAB_ASCII, est codée B8 70 00 puisque (DS) a été défini par AT 70H ;
— MOV CX, LENGTH TAB_REG, codée B9 10 00 met la longueur 10H de TAB_REG dans CX dans ce cas éviter de fractionner la chaîne de caractères par un retour chariot sans faire précéder la nouvelle ligne par & ;
— LEA BX, TAB_REG, codée 8D 1E 9F 00 met 009F dans BX... comme
MOV BX, OFFSET TAB_REG ;
— LDS BX, DWORD PTR TAB_REG, codée C5 1E 9F 00 charge BX et DS. BX est chargé avec le contenu des deux cases mémoires (8 bits) pointées par TAB_REG et DS est chargé avec le contenu des deux cases mémoires pointées par TAB_REG + 2, nous utilisons donc deux mots de 16 bits, c'est pourquoi il faut préciser que nous changeons l'attribut de TAB_REG par :

DWORD PTR

```
MCS-86 MACRO ASSEMBLER    ESSAI

ISIS-II MCS-86 MACRO ASSEMBLER V2.1 ASSEMBLY OF MODULE ESSAI
OBJECT MODULE PLACED IN :F1:ESSAI.OBJ
ASSEMBLER INVOKED BY:  ASM86 :F1:ESSAI.SRC DEBUG SYMBOLS

LOC  OBJ                  LINE     SOURCE

----                        1      TABLES   SEGMENT AT 70H
                            2
008F                        3               ORG      8FH
                            4
                            5
008F 30313233343536         6      TAB_ASCII  DB   '0123456789ABCDEF'
     37383941424344
     4546
                            7
009F 41584258435844         8      TAB_REG    DB   'AXBXCXDXSIDIBPSP'
     58534944494250
     5350
                            9
----                       10      TABLES   ENDS
                           11
----                       12      CODESEG SEGMENT
                           13
                           14      ASSUME   CS:CODESEG,DS:TABLES
                           15
0000 A18F00                16      MOV      AX,WORD PTR TAB_ASCII
                           17
0003 8A1E8F00              18      MOV      BL,TAB_ASCII
                           19
0007 BB9F00                20      MOV      BX,OFFSET TAB_REG
                           21
000A B87000                22      MOV      AX,SEG TAB_ASCII
                           23
000D B91000                24      MOV      CX,LENGTH TAB_REG
                           25
0010 8D1E9F00              26      LEA      BX,TAB_REG
                           27
0014 C51E9F00              28      LDS      BX,DWORD PTR TAB_REG
                           29
----                       30      CODESEG ENDS
                           31               END
```

Exemple 2

A l'intérieur d'un segment, la directive DD peut être utilisée pour charger l'offset et le segment d'une variable, ainsi :

```
TABLES      SEGMENT
TAB1        DB          10   DUP (?)
TAB2        DB          5    DUP (?)
AD__TAB1    DD          TAB1
AD__TAB2    DD          TAB2
TABLES      ENDS
```

réserve :

— 10 octets **indéfinis — DUP (?) — à partir d'un OFFSET 0**
— 5 mots (10 octets) à partir d'un OFFSET 0AH
— range en mémoire aux adresses relatives 20D (14H) et 21D l'OFFSET de TAB1 soit 0000 et en 22D et 23D la valeur du segment TABLES, en 24D et 25D l'OFFSET de TAB2 soit 000AH et en 26D et 27D la valeur du segment TABLES.

Ceci permet de charger en cours de programme un registre et un registre de segment, simultanément, par exemple :

```
LDS     BX, AD_TAB2
```

charge :

BX avec l'OFFSET de TAB2 (000AH)
et DS avec le SEGMENT de TAB2

II-7. La directive ASSUME

Nous avons donné des noms aux segments afin de clarifier la lecture du programme. Il faut maintenant, pour la génération des codes, préciser quels sont les registres de segment. Cette opération est dévolue à la directive ASSUME que l'on écrit généralement en première ligne dans le segment des codes mais qui peut être mise n'importe où. Par exemple (exemples 1 et 2) :

```
TABLES          SEGMENT
TAB_ASCII       DB      '0123456789ABCDEF'
TAB_REG         DB      'AXBXCXDXSIDIBPIP'
TABLES          ENDS
CODESEG         SEGMENT
ASSUME          CS : CODESEG, DS : TABLES
                :
CODESEG         ENDS
```

Si nous avons écrit des données dans le segment appelé CODESEG, certaines instructions nécessiteront un préfixe de changement de segment d'adressage. Par exemple :

```
CODE            SEGMENT
ASSUME          CS : CODE, DS : DATA
                :
TAB_ASCII       DB              '0123456789ABCDEF'
                :
                LEA             BX, TAB_ASCII
                :
                XLAT            TAB_ASCII

                :
CODE            ENDS
```

génère une erreur pour XLAT car cette instruction utilise (BX) pour l'adressage donc le segment DS mais TAB_ASCII est définie par rapport à CS.

L'assembleur ASM86 qui a la première passe a codé XLAT D7, s'aperçoit à la deuxième passe qu'il faut préfixer XLAT mais n'a pas la place de le faire. Il faut écrire :

XLAT CS : TAB_ASCII

Cette réservation d'octets à la première passe conduit parfois à la génération de l'instruction NOP codée 90. C'est en particulier le cas pour les sauts courts vers une adresse plus élevée, à moins qu'on ne l'ait précisé, par **SHORT.**

La directive ASSUME demeure effective jusqu'au nouvel ASSUME. L'absence de cette directive est équivalente à :

ASSUME NOTHING

dans ce cas, les codes dépendant de registres de segments ne sont pas générés puisque les segments ne contiennent rien. NOTHING peut être réservé à un seul segment et doit être utilisé avant un changement d'affectation.

II-8. La directive GROUP

Cette directive permet le regroupement de plusieurs segments de noms différents pour leur attribuer le même registre de segment. Par exemple :

```
DATAGRP    GROUP DATA1, DATA2
DATA1      SEGMENT
COMPTE     DW    ----
:
DATA1      ENDS
DATA2      SEGMENT
:
DATA2      ENDS
:
ASSUME     DS : DATAGRP
```

Dans ce cas, si l'on utilise OFFSET avec une variable appartenant à un groupe, il faut utiliser le nom du groupe en préfixe :

MOV BX, OFFSET DATAGRP : COMPTE

III. LES OPERATEURS PTR, SHORT, THIS

III-1. PTR permet de préciser la taille du mot concerné lors de traitement de cases mémoires avec adressage indirect :

— opérations arithmétiques ou logiques immédiates,

— sauts ou appels indirects.
On écrit

```
MOV   [DI], BL
```

mais il faut écrire

```
MOV   BYTE PTR [DI], 7AH
```

ou

```
MOV   WORD PTR [DI], 7AH
```

De même, il faut écrire

```
JMP   WORD PTR [BX]
```

si l'on désire un saut *intrasegment* à partir d'une adresse pointée par (BX) dans le segment DS et

```
JMP   DWORD PTR [BX]
```

s'il s'agit d'un saut **intersegment** avec changement de (CS).

Il existe un cas particulier :

```
PUSH   WORD PTR [DI]
```

PTR est également utilisé pour changer la dimension d'une variable (BYTE, WORD...), nous avons vu :

```
LDS   BX, DWORD PTR TAB_REG
```

On emploie PTR précédé de **NEAR** (intersegment) ou **FAR** (intersegment) avec un saut, un appel, par exemple :

```
JMP   NEAR PTR BRANCHE
```

III-2. SHORT — Nous permet de gagner un octet dans les sauts, inconditionnels, intrasegment vers une instruction définie par un LABEL Mais dont l'adresse est plus haute que celle du saut. Cela évite à ASM86 de prévoir 3 octets. On écrit :

```
JMP   SHORT SUITE
```

III-3. THIS — Alloue à la variable la case mémoire suivante ainsi :

```
MON_OCTET    EQU   THIS BYTE
MON_MOT      DW    ?
```

attribue à MON_OCTET le même OFFSET que MON_MOT mais pour un octet seulement. Cet opérateur est souvent utilisé pour définir le haut d'une pile.

```
STACK          SEGMENT
               DW 24 DUP (?)
HAUT_DE_PILE   EQU THIS WORD
STACK          ENDS
```

On peut également écrire

```
HAUT_DE_PILE   LABEL WORD
```

IV — DIRECTIVE RECORD ET STRUCTURE

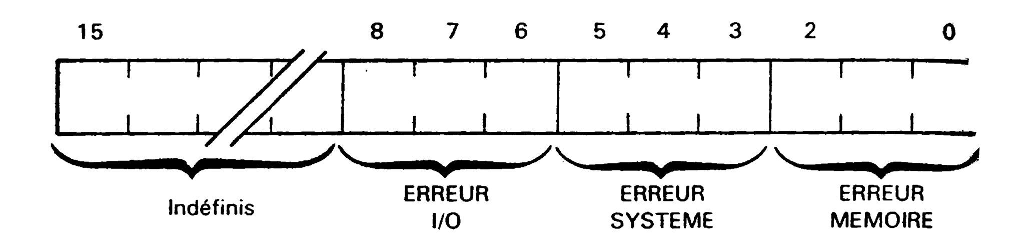
Il s'agit de deux directives permettant de définir l'une un mot en travaillant au niveau du bit (RECORD) l'autre un ensemble de mots.

IV-1. RECORD

Sert à définir des groupes de bits à l'intérieur d'un mot, par exemple :

```
ERREUR_FLAGS  RECORD
&             IOERR : 3
&             SYSTEM_ERR : 3
&             MEMERR : 3
```

donne le mot ERREUR_FLAGS ainsi composé :



Pour initialiser les différents groupes, on écrit par exemple :

```
   FLAG1  ERREUR_FLAGS <0,3,0>
ou FLAG2  ERREUR_FLAGS < , ,2>
```

et en cours de programme :

```
MOV    AX, ERREUR_FLAGS <5,4,3>
```

ce qui donne (AX) = 0163H

Le programme suivant définit un mot EMP_BYTE dont 6 bits indiquent les années de service, un bit le sexe de l'employé(e) et un bit son statut, puis recherche parmi les employées celles qui ont 10 ans et plus de service et qui ne sont pas exemptées. Les masques ont été évidemment définis :

```
MASKSEX      EQU   00000010B ;FEMME
MASKSTATUS   EQU   00000001B ;EXEMPTEE
```

```
          EMP_BYTE DB ?                 ; 1 BYTE, UNINITIALIZED
          ; BIT DEFINITIONS:
          ;  7-2      :YEARS EMPLOYED
               1      :SEX (1=FEMALE)
               0      :STATUS (1=EXEMPT)
          EMP_BITSRECORD                ;RECORD DEFINED HERE
          &           YRS_EMP:6,
          &           SEX:1,
          &           STATUS:1
          .
          .
          .
          ;SELECT NON EXEMPT FEMALES EMPLOYED 10+YEARS

          MOV         AL, EMP_BYTE      ;KEEP ORIGINAL INTACT
          TEST        AL, MASK SEX      ;FEMALE?
          JZ          REJECT            ;NO, QUITE
          TEST        AL, MASK STATUS   ;NON EXEMPT ?
          JNZ         REJECT            ;NO, QUIT
          SHR         AL, CL            ;ISOLATE YEARS
          CMP         AL, 11            ;>=10 YEARS?
          JL          REJECT            ;NO, QUIT
          ;PROCESS SELECTED EMPLOYEE
          .
          .
          .
REJECT:   ;PROCESS REJECTED EMPLOYEE
          .
          .
          .                             ;RECORD USED HERE
          MOV         CL, YRS_EMP       ;GET SHIFT COUNT
```

Remarquez & et, dans la définition de EMP_BITSRECORD.

IV-2. STRUCTURE

Permet de définir un groupe d'octets. Par exemple :

```
EMPLOYE            STRUC
EMP_NOM            DB '                 ;20 CARACTERES AUTORISES
SAL_HEURE          DD 26.40
NOMBRE_HEURES      DB ?
EMPLOYE            ENDS
```

Les initialisations pourront être

```
DUPONT EMPLOYE <'DUPONT, JULES', 28.95, 60>
```

ou

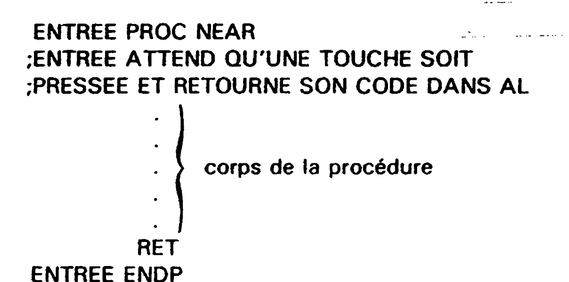
```
PERSONNEL EMPLOYE 20 DUP (<>)
```

Pour accéder aux différents octets d'une structure :

```
MOV   AX, OFFSET DUPONT
MOV   CX, DUPONT.NOMBRE__HEURES
```

V. PROC

Cette directive permet de définir un label et un ensemble d'instructions qui est généralement interprété comme un sous-programme, qui doit être défini comme appartenant ou non au même segment que le programme appelant afin que les retours et les appels soient correctement codés. Si le sous-programme est dans le même segment que le programme appelant, il est dit NEAR sinon il est FAR. Exemple :



```
ENTREE PROC NEAR
;ENTREE ATTEND QU'UNE TOUCHE SOIT
;PRESSEE ET RETOURNE SON CODE DANS AL
          .
          .
          .   } corps de la procédure
          .
          .
        RET
ENTREE ENDP
```

Le retour de ENTREE est codé C3 et un appel à ENTREE sera codé E8... puisqu'il s'effectuera sans changement de (CS). Il ne faut pas oublier de signaler la fin de la procédure par son nom suivi de ENDP. L'exemple suivant allie la notion de structure servant ici à un passage de paramètres à la notion de procédure, il faut remarquer :

— la séquence

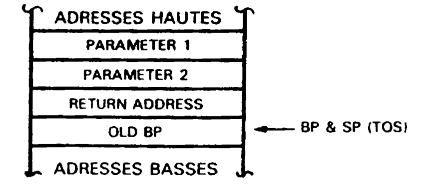
```
PUSH BP
MOV  BP, SP
```

qui permet l'adressage via (BP) et que l'on retrouve en PLM86 et dans l'instruction ENTER ;

— l'adressage des octets d'une structure

```
MOV   AX, [BP].PARM__1
```

L'arrangement de la structure est donnée dans le schéma ci-dessous.



```
CODE            SEGMENT
                ASSUME CS:CODE
MAX               PROC
;THIS PROCEDURE IS CALLED BY THE FOLLOWING
;     SEQUENCE:
;          PUSH PARM1
;          PUSH PARM2
;          CALL MAX
;IT RETURNS THE MAXIMUM OF THE TWO WORD
;  PARAMETERS IN AX.

;DEFINE THE STACK LAYOUT AS A STRUCTURE
STACK__LAYOUT STRUC
OLD__BP          DW ?    ;SAVED BP VALUE__BASE OF STRUCTURE
RETURN__ADDR     DW ?    ;RETURN ADDRESS
PARM__2          DW ?    ;SECOND PARAMETER
PARM__1          DW ?    ;FIRST PARAMETER
STACK__LAYOUT    ENDS

;PROLOG
                 PUSH   BP                   ;SAVE IN OLD__BP
                 MOV    BP, SP               ;POINT TO OLD__BP
;BODY
                 MOV    AX, [BP].PARM__1     ;IF FIRST
                 CMP    AX, [BP].PARM__2     ;>SECOND
                 JG     FIRST__IS__MAX       ;THEN RETURN FIRST
                 MOV    AX, [BP].PARM__2     ;ELSE RETURN SECOND
;EPILOG
FIRST__IS__MAX:  POP    BP                   ;RESTORE BP (& SP)
;RETURN
                 RET    4                    ;DISCARD PARAMETERS
MAX              ENDP
CODE             ENDS
                 END
```

VI. PUBLIC ET EXTRN

Ces deux directives sont utilisées lors de la réunion de plusieurs programmes par LINK86. PUBLIC précise que les symboles utilisés (variables, étiquettes, constantes) sont valables pour tous les programmes.

EXTRN précise que les symboles utilisés dans le programme ont été définis par ailleurs et déclarés PUBLIC.
Par exemple :

```
DATA      SEGMENT
          PUBLIC TAMPON             ;PREMIERE DEFINITION
TAMPON    DB 100 DUP (?)
             :
CODE      ENDS
```

autre programme, d'un autre utilisateur ayant accès à la bibliothèque du précédent.

```
        EXTRN TAMPON:BYTE
CODE    SEGMENT
        ASSUME CS:CODE,...
          .
          .
          .
CODE    ENDS
        END
```

Ici, on utilise TAMPON qui a été défini ailleurs comme PUBLIC et comme un ensemble d'octets, on rappelle son TYPE (:BYTE). Si le premier utilisateur n'avait pas déclaré TAMPON comme PUBLIC, le deuxième utilisateur n'y aurait pas eu accès.

VII . MACRO

Nous donnons ici le principe de l'écriture d'une MACRO pour plus de détails, reportez-vous aux ouvrages INTEL.

Pour définir une macro, on utilise la syntaxe suivante, en début de *programme principal.*

```
%*DEFINE (nom de la macro)(
         :  )
         :  } instructions de la macro
         :  )
         )
```

Exemple :

```
%*DEFINE (ADD5)(
MOV     CX,100
MOV     SI,0
MOV     AX,TABLE2[SI]
ADD     AX,5
MOV     TABLE2[SI],AX
INC     SI
INC     SI
LOOPZ   $-13
)
```

Dans le programme principal, nous aurons :

```
PUSH  AX
%ADD  5
POP   AX
```

Le programme après assemblage sera :

```
PUSH   AX
MOV    CX,100
MOV    SI,0
MOV    AX,TABLE2[SI]
ADD    AX,5
MOV    TABLE2[SI],AX
INC    SI
INC    SI
LOOPZ  $-13
POP    AX
```

L'exemple suivant montre qu'une macro peut faire appel à une macro, avec :

```
%*DEFINE (MOVE)(
MOV    CX,100
LEA    SI,TABLE1
LEA    DI,TABLE2
REP    MOVSW
)
```

Nous écrirons :

```
%*DEFINE (MOV_AND_ADD)(
%MOVE
%ADD5
)
```

La macro sera appelée dans le programme principal :

```
%MOVE_AND_ADD
```

Le programme après assemblage sera :

```
MOV    CX,100
LEA    SI,TABLE1
LEA    DI,TABLE2
REP    MOVSW
MOV    CX,100
MOV    SI,0
MOV    AX,TABLE2[SI]
ADD    AX,5
MOV    TABLE2[SI],AX
INC    SI
INC    SI
LOOPZ  $_13
```

Une macro peut avoir des paramètres :

```
%*DEFINE (MOV_ADD_GEN(SOURCE, DEST, NBRE)(
        MOV     CX,%NBRE
        MOV     SI,Ø
   @Ø:  MOV     AX, %SOURCE [SI]
        MOV     %DEST [SI], AX
        INC     SI
        INC     SI
        LOOPZ   @Ø
        )
```

Dans le programme principal, nous écrirons :

```
% MOV_ADD_GEN (INPUT, STORE, 100/H)
```

ce qui donnera :

```
        MOV     CX, 100H
        MOV     SI, Ø
   @Ø:  MOV     AX, INPUT [SI]
        MOV     STORE [SI], AX
        INC     SI
        INC     SI
        LOOPZ   @Ø
```

Afin d'éviter des erreurs dues aux variations du nombre d'octets du code des instructions, nous remplaçons l'étiquette @ Ø : par un *label local*, c'est-à-dire :

```
%LABEL:   MOV AX, % SOURCE [SI]
          .
          .
          .
          .
          .
          LOOPZ  %LABEL
```

VIII. ASSEMBLAGE CONDITIONNEL

En assemblage conditionnel, on peut utiliser les opérateurs suivants :

EQ	égal à	*(Equal)*
NE	différent de	*(Not Equal)*
LT	plus petit que	*(Lower Than)*
GT	plus grand que	*(Greater Than)*
LE	plus petit que ou égal à	*(Lower or Equal)*
GE	plus grand que ou égal à	*(Greater or Equal)*

qui ont pour effet de porter à Ø une valeur tampon si la relation est *vraie* et à ØFFFFH si la relation est fausse. Ainsi :

```
MOV BX, ((POR_VAL LT 5) AND 20)
& OR ((POR_VAL GE 5) AND 30)
```

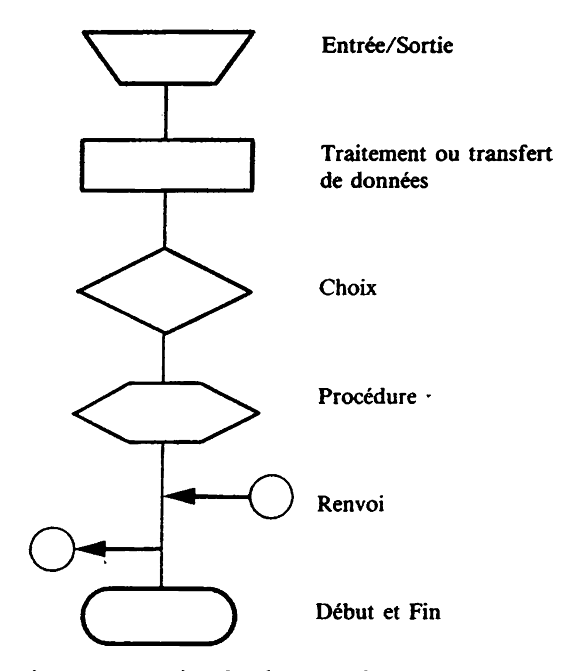
donne MOV BX, 20 si POR_VAL est plus petit que 5 et MOV BX,30 si POR_VAL est supérieur ou égal à 5.

Si vous utilisez de telles formules, faites bien attention aux parenthèses.

IX. EXEMPLES

Les exemples sont en partie extraits des ouvrages INTEL mais avant de les examiner, nous donnons quelques conseils aux débutants.

1) Etablissez l'organigramme du travail à effectuer en essayant d'être le plus clair et le plus simple possible à l'aide des schémas conventionnels.



2) Ne pas envisager au premier abord un grand programme.

3) Fractionner le travail à effectuer en « modules ». Mettre au point chaque module puis les réunir en prenant garde :

— au transfert des paramètres

— aux contenus des registres.

Un programme réalisé avec beaucoup de « Procédures » est très clair mais il est plus lent qu'un programme « linéaire » mais ce dernier est plus délicat à mettre au point.

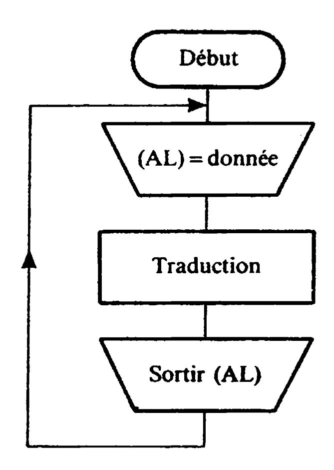
4) Lors de la mise au point des modules, ne pas restreindre les essais à une valeur de la variable. Il faudra se méfier surtout des cas limites (erreur dans l'écriture des comparaisons et des conditions de sauts).

Exemple 1

Traduit la valeur reçue du port 1 en code Gray pour l'émettre par le port 2 (USART).

```
MY_DATA SEGMENT
GRAY        DB          18H,34H,05H,06H,09H,0AH,0CH,11H,12H,14H
MY_DATA ENDS

MY_CODE SEGMENT
            ASSUME      CS:MY_CODE, DS:MY_DATA
GO:         MOV         AX,MY_DATA          ;initialisation de (DS)
            MOV         DS,AX
            MOV         BX,OFFSET GRAY      ;chargement de BX
CYCLE:      IN          AL,1                ;lecture de la donnée
            XLAT        GRAY                ;traduction
            OUT         2,AL                ;émission
            JMP         CYCLE               ;répétition
MY_CODE ENDS
            END         GO
```



Si la table GRAY est écrite dans MY_CODE, il faut écrire XLAT CS:GRAY.

Exemple 2

Réalise l'addition décimale de 2 nombres entrés en codes ASCII et rangés en mémoire à partir de STRING_1 et STRING_2, le résultat prend la place du deuxième nombre. Le nombre de chiffres est chargé dans CX, qui sert de compteur de boucle.

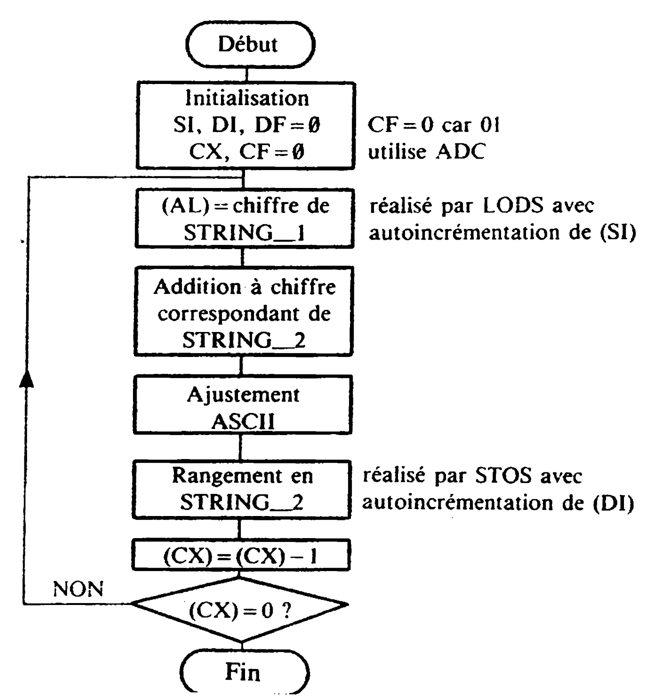
```
MY_DATA SEGMENT
STRING_1 DB         '1','7','5','2'         ;soit 2571
STRING_2 DB         '3','8','1','4'         ;soit 4183
MY_DATA ENDS

MY_CODE SEGMENT
         ASSUME     CS:MY_CODE, DS:MY_DATA
GO:      MOV        AX,MY_DATA              ;initialisation de (DS)

         MOV        DS,AX

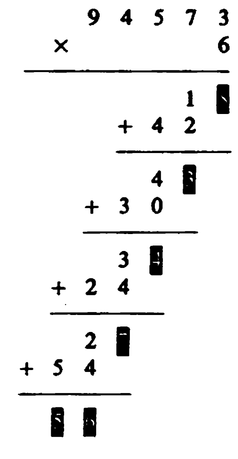
         MOV        CX, LENGTH STRING_1 ;initialisation de (CX)
         CLC                                ;CF=0
         CLD                                ;auto incrémentation de
                                            ;SI et DI

         MOV        SI,OFFSET STRING_1      ;initialisation de (SI) et (DI)
         MOV        DI,OFFSET STRING_2
CYCLE:   LODS       STRING_1                ;(AL)=chiffre de STRING_1
         ADC        AL,[DI]                 ;ajouter à chiffre de STRING_2
         AAA                                ;correction ASCII
         STOS       STRING_2                ;rangement
         DEC CX
         JNZ        CYCLE                   ;répétition
         HLT                                DEC CX
MY_CODE ENDS
         END        GO
```



Exemple 3

Réalise la multiplication décimale d'un nombre de plusieurs chiffres entrés un à un en ASCII par un chiffre ASCII.
Le nombre commence en 'A', le résultat est rangé à partir de 'C'.
L'opération s'effectue comme nous avons l'habitude de la faire. A savoir :



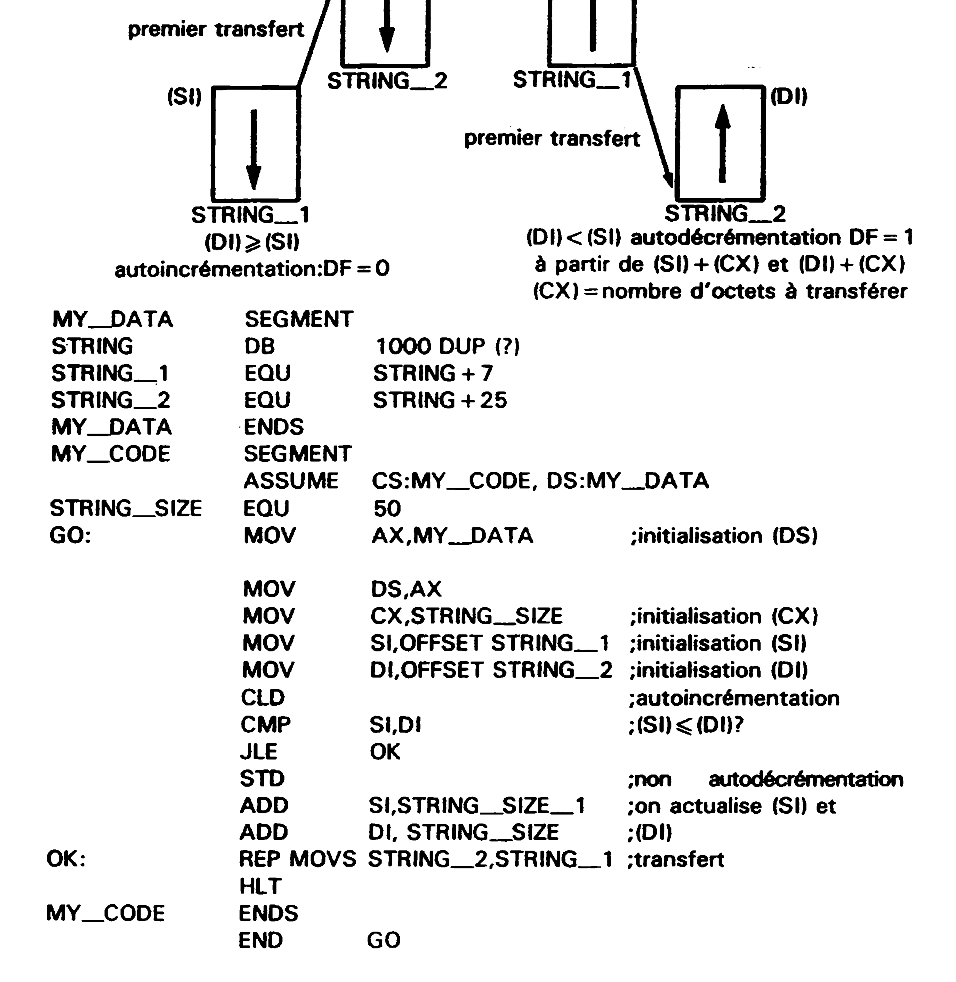
```
MY_DATA SEGMENT
A        DB       '3','7','5','4','9'
B        DB       '6'
C        DB       LENGTH (A) DUP (?)
MY_DATA ENDS
MY_CODE SEGMENT
         ASSUME   CS:MY_CODE,DS:MY_DATA
GO:      MOV      AX,MY_DATA          ;initialisation de (DS)
         MOV      DS,AX
         CLD                          ;autoincrémentation
         MOV      SI,OFFSET A         ;initialisation de (SI)
         MOV      DI,OFFSET C         ;initialisation de (DI)
         MOV      CX,LENGTH A         ;initialisation de (CX)
         AND      B,0FH               ;octet fort de B=0
         MOV      BYTE PTR [DI],0     ;résultat=0
CYCLE:   LODS     A                   ;(A)=1er chiffre avec (SI)=(SI)+1
         AND      AL,0FH
         MUL      AL,B                ;multiplication
         AAM                          ;correction ASCII
         ADD      [DI]                ;addition au résultat précédent
         AAA                          ;correction ASCII
         STOS     C                   ;rangement avec (DI)=(DI)+1
         MOV      [DI],AH             ;rangement des dizaines
         DEC      CX
         JNZ      CYCLE               ;répétition
         HLT
MY_CODE ENDS
         END      GO
```

On remarquera l'utilisation de :
— SI et DI avec autoincrémentation grâce à LODS et STOS
— BYTE PTR [DI],0

Quel résultat aurions-nous si nous rangions les dizaines MOV[DI],AH avant de stocker les unités STOS C ?

Exemple 4

Transfert 50 octets entre deux suites qui se recouvrent.
En raison du recouvrement, la valeur de DF est ajustée pour opérer le transfert dans l'ordre adéquat :



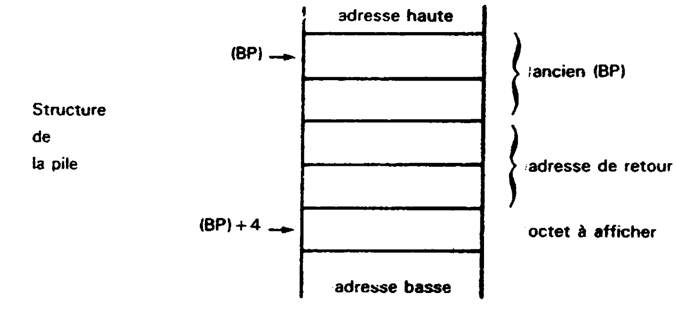

```
MY__DATA       SEGMENT
STRING         DB         1000 DUP (?)
STRING__1      EQU        STRING+7
STRING__2      EQU        STRING+25
MY__DATA       ENDS
MY__CODE       SEGMENT
               ASSUME     CS:MY__CODE, DS:MY__DATA
STRING__SIZE   EQU        50
GO:            MOV        AX,MY__DATA          ;initialisation (DS)

               MOV        DS,AX
               MOV        CX,STRING__SIZE      ;initialisation (CX)
               MOV        SI,OFFSET STRING__1  ;initialisation (SI)
               MOV        DI,OFFSET STRING__2  ;initialisation (DI)
               CLD                             ;autoincrémentation
               CMP        SI,DI                ;(SI)≤(DI)?
               JLE        OK
               STD                             ;non    autodécrémentation
               ADD        SI,STRING__SIZE__1   ;on actualise (SI) et
               ADD        DI, STRING__SIZE     ;(DI)
OK:            REP MOVS STRING__2,STRING__1    ;transfert
               HLT
MY__CODE       ENDS
               END        GO
```

Exemple 5

Il s'agit d'un sous-programme qui affiche un octet préalablement sauvé en pile par un PUSH. Ici, la table ASCII (TAB_ASCII) est dans le segment des codes. SORTIE émet l'octet rangé en CHAR.

```
AFF1:   LEA     BX,TAB_ASCII        ;initialisation de BX
        PUSH    BP
        MOV     BP,SP               ;BP sert de pointeur en pile
        MOV     AL,[BP+4]           ;chargement de l'octet
        AND     AL,0F0H             ;on garde les quatre bits
        MOV     CL,04               ;de poids fort
        ROL     AL,CL
        XLAT    CS:TAB_ASCII        ;traduction ASCII
        MOV     CHAR,AL
        CALL    SORTIE
        MOV     AL,[BP+4]
        AND     AL,0FH              ;quatre bits de poids faible
        XLAT    CS:TAB_ASCII
        MOV     CHAR,AL
        CALL    SORTIE
        POP     BP
        RET     2                   ;« vidange » de la pile
```



Exemple 6

Ce programme affiche un ensemble d'octets séparés par des espaces, le premier est pointé par (SI) le dernier par (DI). Il utilise le programme précédent, c'est pourquoi on peut lire :

PUSH WORD PTR [SI]

Le programme appelé ESP sort un espace, MEM_IP est une case mémoire tampon on sort avec (SI) = (DI).

```
AFF_CODE:
          MOV     CX,DI
          SUB     CX,SI
          INC     CX
          ADD     MEM_IP,CX
          PUSH    CX
CONT:     PUSH    WORD PTR [SI]
          CALL    AFF1
          CALL    ESP
          POP     CX
          INC     SI
          LOOP    CONT
          INC     DI
          RET
```

Exemple 7

Ce programme permet de s'aiguiller vers différents programmes suivant une touche pressée. Il faut, au préalable, charger BX avec l'adresse du code de la touche *moins* 3 et CX au nombre de touches possibles. La table pointée par BX contient les codes des touches suivis de l'adresse du programme concerné :

```
'A'
PROG_A
'B'
PROG_B
  .
  .
  .
```

Remarquez : JMP WORD PTR CS:[BX] et CMP AL,CS:[BX]

car la table est dans le segment des codes.

```
BRANCH1:
          ADD     BX,03
          CMP     AL,CS:[BX]
          LOOPNZ  BRANCH1
          JNZ     ERREURO
BRANCHO:
          INC     BX
          JMP     WORD PTR CS:[BX]
ERREURO:                              ;émet un message d'erreur
```

Exemple 8

Il s'agit du début d'un programme nous donnant des exemples d'emploi de différentes directives.

```
MCS-86 MACRO ASSEMBLER ASS_SDK86

ISIS-II MCS-86 MACRO ASSEMBLER V2.1 ASSEMBLY OF MODULE ASS_SDK86

OBJECT MODULE PLACED IN :F1:BG86.OBJ
ASSEMBLER INVOKED BY: ASM86 :F1:BG86.SRC ERRORPRINT

LOC OBJ       LINE   SOURCE

                1    $DEBUG
                2
                3    NAME       ASS_SDK86
                4
                5+1  $TITLE('ASSEMBLEUR SDK 86 -VER 1- 10/83 ')
                6
                7
------          8    DATASEG   SEGMENT
                9
0000 (102      10               NONACCES   DB          66H DUP(?)
     ??
     )
0066 ??        11               CHAR       DB          ?
0067 ????      12               MEM_IP     DW          ?
               13
------         14    DATASEG   ENDS
               15
               16
------         17    STACKSEG              SEGMENT
               18
0000 (128      19               DW         80H DUP(?)
     ????
     )
               20
               21
------         22    STACKSEG              ENDS
               23
               24
------         25    CODESEG               SEGMENT
               26
               27    ASSUME    CS:CODESEG, DS:DATASEG, SS:STACKSEG
               28
0000           29    ENTREE    PROC        NEAR
               30
0000 52        31              PUSH        DX
0001 BAF2FF    32              MOV         DX,0FFF2H    ;INIT 8251
0004 EC        33    @0:       IN          AL,DX
0005 A802      34              TEST        AL,02        ;RX READY
0007 74FB      35              JZ          @0
0009 B2F0      36              MOV         DL,0F0H
000B EC        37              IN          AL,DX
000C 247F      38              AND         AL,7FH
```

```
000E EE        ECHO:     OUT       DX,AL
000F 5A                  POP       DX
0010 C3                  RET

0011 52        SORTIE:   PUSH      DX
0012 BAF2FF              MOV       DX,0FFF2H
0015           PAS_PRE-  T
                         :
0015 EC                  IN        AL,DX
```

Exemple 9 :

Ce programme vous précise l'utilisation de REPNZ. En effet, la comparaison du contenu de AL, ici 0D, code ASCII du *retour chariot,* aux différents octets de la suite de caractères pointés par (DI) est répétée **jusqu'à** ce que le résultat soit nul (égalité) ou que (CX) soit nul, mais nous sommes sûrs que 0D existe à la fin de chaque mot ce qui rend l'initialisation de CX inutile.

On écrit SCAS LISTE afin de générer correctement le code de SCAS. LISTE a été définie comme une suite d'octets (DB) dans un segment défini par ES grâce à ASSUME. Si le segment de LISTE avait été DS, par exemple, nous aurions eu un message d'erreur car les instructions traitant des chaînes de caractères pointées par (DI) ne peuvent être « préfixées » (overriden).

```
LOC  OBJ        SOURCE

                ;RECHERCHE L'ADRESSE DU DERNIER CARACTERE
                ;D'UNE SUITE DE MOTS DE LONGUEUR VARIABLE
                ;LE DERNIER CARACTERE DE CHAQUE MOT EST OD
                ;
                ;HYPOTHESES: (DI)=ADRESSE PREMIERE LETTRE
                ;                 DE LA SUITE
                ;            (DX)=NOMBRE DE MOTS DE LA LISTE
                ;
----            DICTIONNAIRE     SEGMENT
                ;
0000 (1000      LISTE     DB     1000 DUP(?)
     ??
     )
                ;
----            DICTIONNAIRE     ENDS
                ;
----            RECHERCHE        SEGMENT
                ;
  0700          AD_LET           EQU     0700H
  0060          NBRE_MOTS        EQU     060H
                ;
                ASSUME CS:RECHERCHE,ES:DICTIONNAIRE

                ;
0000 BF0007               MOV    DI,AD_LET
0003 BA6000               MOV    DX,NBRE_MOTS
0006 B00D                 MOV    AL,ODH
```

```
0008            CONTINUE:
0008 F2                   REPNZ   SCAS LISTE
0009 AE
000A 4A                   DEC     DX
000B 75FB                 JNZ     CONTINUE
                ;
                ;
                ;
                ;
                ;ON SORT DE LA BOUCLE AVEC (DI) POINTANT LE DERNIER
                ;CARACTERE + 1
                ;
                ;
----            RECHERCHE         ENDS
                ;
                END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Exemple 10 :

La recherche d'un mot exige un travail octet par octet même avec des processeurs traitant des mots de plusieurs octets, en particulier dans le cas de recherches alphabétiques.

Ici après REPZ on trouve JCXZ, car on sort de la boucle de répétition si le contenu de CX est nul (toutes les lettres examinées) d'où JCXZ ou si une lettre du mot n'est pas identique à une lettre de même place d'un mot de la liste (REPZ).

Dans le dernier cas, CX contient le nombre de lettres restant à examiner ce qui justifie ADD DI, CX pour que (DI) pointe la première du mot suivant dans la liste, puis ADD DI, LONG-INF pour pointer la première lettre du premier mot du fichier suivant. Remarquez l'emploi de OFFSET, LENGTH.

```
LOC   OBJ        SOURCE

                 ;RECHERCHE UN MOT DANS UNE LISTE,CE MOT
                 ;DEBUT D'UN FICHIER EST DE LONGUEUR FIXE
                 ;
                 ;LE MOT ENTRE EST POINTE PAR (SI),LA LISTE
                 ;PAR (DI)
                 ;
----             TAMPON  SEGMENT
                 ;
0700                     ORG     0700H
                 ;
0700  (16        MOT     DB      10H DUP(?)
      ??
      )
                 ;
----             TAMPON  ENDS
                 ;
----             FICHIER SEGMENT
                 ;
1000                     ORG     1000H
                 ;
```

```
1000 (256            19     DEB_FICHIER      DB       100H DUP(?)
     ??
     )
  0050               20     LONG_INF         EQU      80D
                     21     ;
----                 22     FICHIER ENDS
                     23     ;
                     24     ;
----                 25     RECHERCHE        SEGMENT
                     26     ;
  0005               27     NBR       EQU    05       ;NOMBRE DE MOTS
                     28     ;
                     29     ASSUME    CS:RECHERCHE,DS:TAMPON,ES:FICHIER
                     30     ;
                     31     ;
0000 BF0010          32               MOV    DI,OFFSET DEB_FICHIER
0003 B305            33               MOV    BL,NBR
0005 FC              34               CLD
0006 B91000          35     RECH:     MOV    CX,LENGTH MOT
0009 BE0007          36               MOV    SI,OFFSET MOT
000C F3              37               REPZ   CMPS MOT,DEB_FICHIER
000D A6
000E E30C            38               JCXZ   TROUVE
0010 03F9            39               ADD    DI,CX
0012 83C750          40               ADD    DI,LONG_INF
0015 FECB            41               DEC    BL
0017 75ED            42               JNZ    RECH
0019 E80300          43               CALL   MESSAGE
001C E8C800          44     TROUVE:   CALL   AFFICHAGE
                     45     ;
```

```
MCS-86 MACRO ASSEMBLER     TABLE


LOC  OBJ                    LINE      SOURCE

                     46     ;
001F                 47     MESSAGE PROC NEAR
                     48     ;
001F (200            49               CARACTERE DB 200 DUP(?)
     ??
     )
                     50     ;
                     51     MESSAGE ENDP
                     52     ;
                     53     ;
00E7                 54     AFFICHAGE PROC NEAR
                     55     AFFICHAGE ENDP
                     56     ;
----                 57     RECHERCHE        ENDS
                     58     ;
                     59     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Exemple 11 :

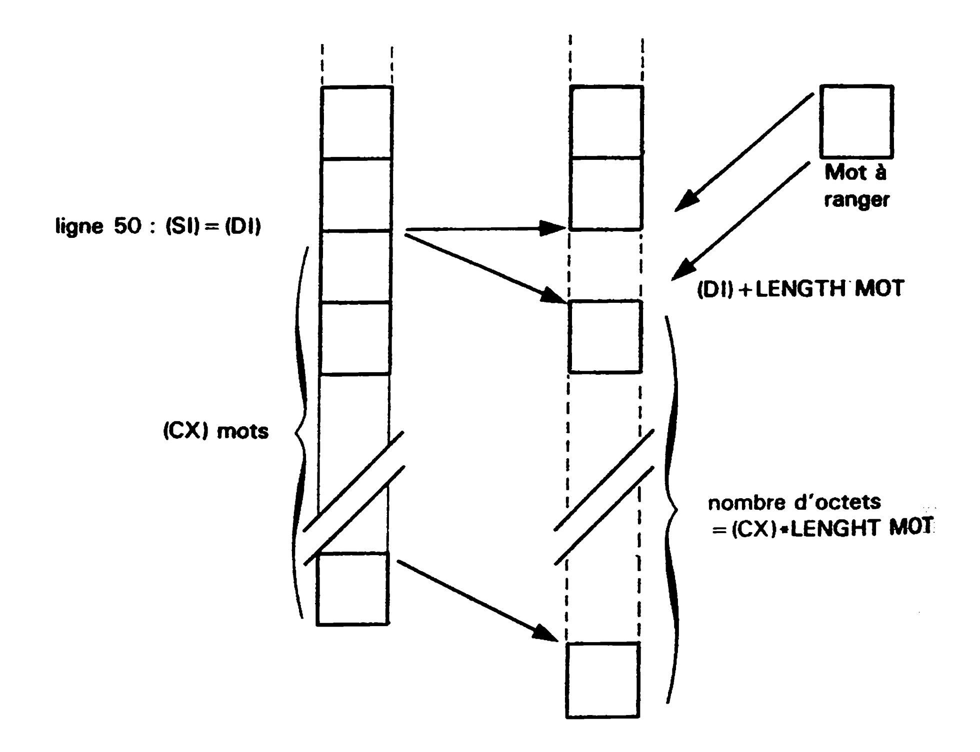
Le rangement par ordre alphabétique fait intervenir JBE (saut si plus petit que ou égal à) qui permet de situer une lettre par rapport aux autres déjà classées d'où l'appellation de REPERE (repéré) qui se précise par le cas de non-égalité en NOUVEAU sinon on passe à la lettre suivante. Par contre, si l'on passe JBE, on doit tester avec une lettre du mot suivant. Par exemple :

BAC
BAL
BAS
BAT

donne l'égalité pour B et A et la non égalité pour la 3e lettre si l'on cherche à classer BAR. On sortira de la répétition après L et avant S sans égalité d'où un branchement à NOUVEAU.

Remarquez le codage de JMP RANGE faisant apparaître un NOP (90). Cette instruction branche directement au rangement du nouveau mot puisqu'il se place à la fin.

Pour mettre le mot en place, il faut décaller les autres, dans l'exemple ci-dessous BAS et BAT pour insérer BAR. CX contient le nombre de mots qui n'ont pas été testés. Le schéma de rangement est le suivant :



Le transfert a lieu par auto-décrémentation de (SI) et (DI) — listes se recouvrant — d'où STD.

Puis on range le nouveau mot dans la place libérée, par auto-incrémentation des anciennes valeurs de (SI) et (DI), d'où CLD, préalablement sauvées à l'aide de PUSH.

Il est plus simple de fixer le nombre de caractères d'un mot (fiche avec cases pour lettres d'imprimerie...) mais cela manque de souplesse ; on peut s'inspirer de l'exemple 9.

```
LOC  OBJ       LINE     SOURCE

                  1     ;RECHERCHE SI UN MOT EXISTE DANS UNE LISTE
                  2     ;ALPHABETIQUE , LE RANGE A SA PLACE S'IL EST
                  3     ;NOUVEAU. LES MOTS SONT DE LONGUEUR FIXE
                  4     ;
----              5     FICHIER SEGMENT
                  6     ;
0700              7             ORG     700H
                  8     ;
0700 (256         9     DEB_FICHIER     DB      100H DUP(?)
     ??
     )
                 10
                 11     ;
----             12     FICHIER ENDS
                 13
----             14     ENTREE  SEGMENT
                 15     ;
0500             16             ORG     500H
                 17     ;
0500 (16         18     MOT     DB      10H DUP(?)
     ??
     )
                 19     ;
----             20     ENTREE  ENDS
                 21
----             22     DICTIONNAIRE    SEGMENT
                 23     ;
 0005            24     NBR     EQU     05      ;NOMBRE DE MOTS
                 25     ;
                 26     ASSUME  CS:DICTIONNAIRE,DS:ENTREE,ES:FICHIER
                 27     ;
0000 BF0007      28             MOV     DI,OFFSET DEB_FICHIER
0003 BE0005      29             MOV     SI,OFFSET MOT
0006 B90500      30             MOV     CX,NBR
0009 BA1000      31             MOV     DX,LENGTH MOT
000C BB0000      32             MOV     BX,0
000F             33     SUIVANTE:
000F 8A00        34             MOV     AL,[BX+SI]
0011             35     SUIVANT:
0011 3A01        36             CMP     AL,[BX+DI]
0013 7608        37             JBE     REPERE
0015 83C710      38             ADD     DI,LENGTH MOT   ;MOT SUIVANT
0018 E2F7        39             LOOP    SUIVANT
```

```
001A EB2290                 JMP     RANGE
001D              REPERE:
001D 7507                   JNE     NOUVEAU
001F 43                     INC     BX
0020 4A                     DEC     DX
0021 75EC                   JNZ     SUIVANTE
0023 EB1E90                 JMP     EXISTE
```

```
MCS-86 MACRO ASSEMBLER    RANG

LOC  OBJ              LINE     SOURCE

0026                           NOUVEAU:
0026 56                                PUSH    SI
0027 57                                PUSH    DI
0028 8BF7                              MOV     SI,DI
002A 83C710                            ADD     DI,LENGTH MOT
002D FD                                STD
002E B81000                            MOV     AX,LENGTH MOT
0031 F7E1                              MUL     CX
0033 03F8                              ADD     DI,AX
0035 03F0                              ADD     SI,AX
0037 8BC8                              MOV     CX,AX
0039 F3                                REP     MOVSB
003A A4
003B FC                                CLD
003C 5F                                POP     DI
003D 5E                                POP     SI
003E                           RANGE:
003E B91000                            MOV     CX,LENGTH MOT
0041 F3                                REP     MOVSB
0042 A4
0043 E80000                    EXISTE: CALL    MESSAGE
                               ;
                               ;
0046                           MESSAGE PROC NEAR
                               ;
0046 (128                      CARACTERES DB   80H DUP(?)
     ??
     )
                               ;
                               MESSAGE ENDP
                               ;
----                           DICTIONNAIRE    ENDS
                               ;
                               END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

80386

La figure 1 donne le schéma interne de ce processeur 32 bits (bus de données de 32 fils) qui peut indifféremment traiter des mots de 8, 16 ou 32 bits, comprenant toutes les instructions des 8086 / 8088, 186 / 188 et 286. La file d'attente est de 16 octets et peut contenir, en moyenne, 5 instructions.
Il se présente en boîtier carré de 132 broches disposées sur 3 rangées.
L'espace mémoire est de 1 Méga-octet (20 bits) en mode réel (celui du 8086), les instructions traitent des mots de 8 ou 16 bits, à moins qu'elles ne soient « préfixées » pour traiter des mots de 32 bits. C'est le mode de travail à la mise sous tension ou après un reset.
Cet espace est de 4 Giga-octets (32 bits) physiques, 64 Tera-octets (46 bits) virtuels en mode protégé, autorisant le 80386 à gérer 12 000 octets (6 pages) par terrien !
Les bus de données et d'adresses sont démultiplexés, la vitesse de transfert est de 32 M-octets par seconde avec un 80386 à 16 MHz.
L'entrée BS16 permet d'informer le 80386 qu'il s'adresse à une configuration 16 bits.

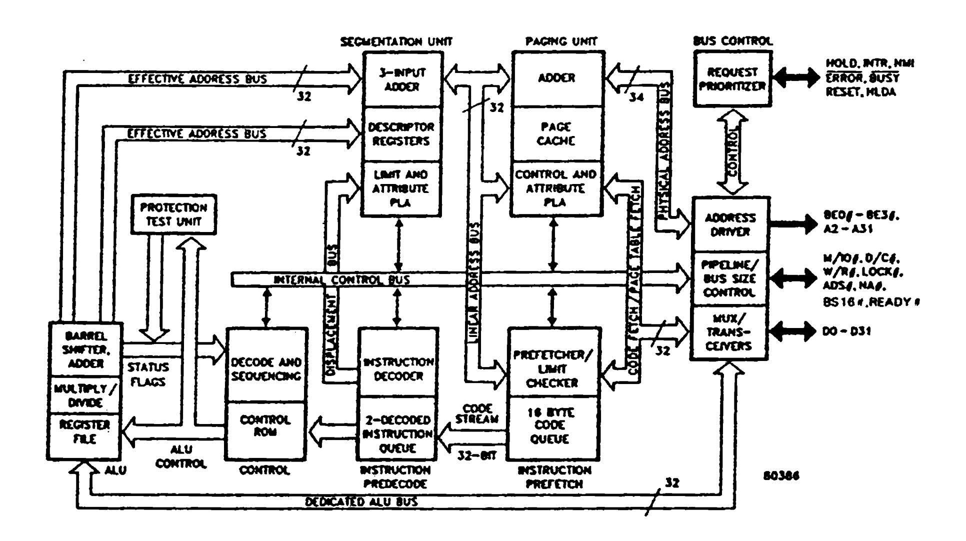


Fig. 1. — Schéma interne du 80386.

I — REGISTRES

Nous retrouvons les registres généraux du 8086 étendus à 32 bits (préfixe E), les registres de segments sont portés à 6 par l'apport de deux segments de données FS et GS. L'ensemble des registres est donné dans le tableau 1.

REGISTRES GENERAUX D'ADRESSES ET DE DONNEES

31 — 16	15 — 8	7 — 0	
	AH A	X AL	EAX
	BH B	X BL	EBX
	CH C	X CL	ECX
	DH D	X DL	EDX
	SI		ESI
	DI		EDI
	BP		EBP
	SP		ESP

REGISTRES DE SEGMENTS

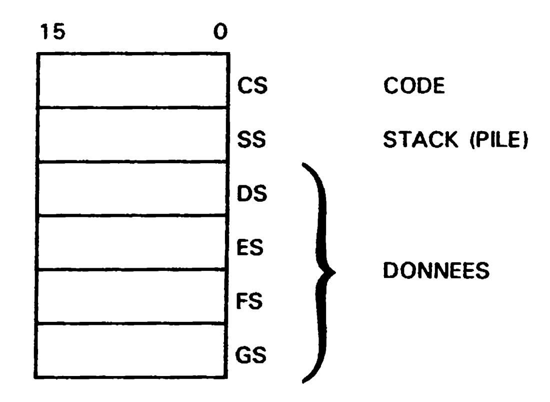


Tableau 1. — Les registres du 80386

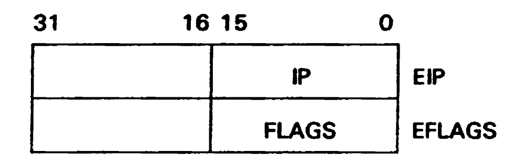


Tableau 1. (suite) — Les registres du 80386.

Le registre des flags étendu, EFLAGS, contient évidemment des flags du 286 (figure 2) complétés par deux nouveaux flags VM et RF.

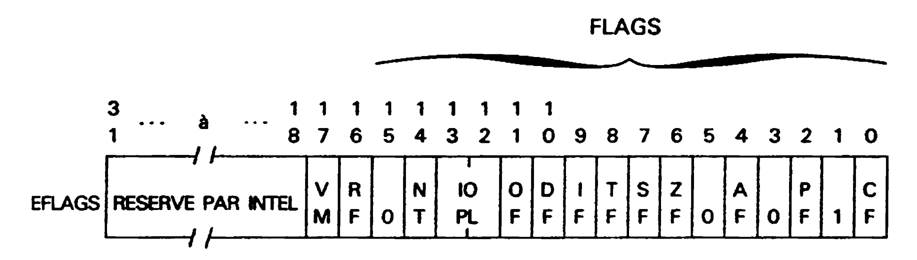


Fig. 2. — Les flags de 80386.

— VM (Mode Virtuel 8086) permet de travailler en mode 8086 alors que le 386 est en mode protégé. Dans ce cas les instructions du mode protégé sont interdites et provoquent une interruption de type 13. Ce flag **n'est pas** affecté par POPF et PUSHF sauve systématiquement un 0 même en mode virtuel 8086.

— RF (Resume Flag) est utilisé conjointement aux registres de points d'arrêt ou de pas à pas. Si RF est à 1, les erreurs sont ignorées lors de la mise au point.

Le mot d'état machine (MSW) du 286 est complété par un bit en poids fort (bit 31) appelé PG (*Paging*) qui, mis à 1, autorise le mode page de 4 en 4 K-octets (12 bits de poids faible de l'adresse basse à 0). Ce registre (MSW plus PG) constitue le Registre 0 de Contrôle de la Machine (CR0). Il est associé à CR2 et CR3, CR1 étant actuellement réservé.

CR2 contient une adresse sur 32 bits correspondant à celle de la dernière page, permettant une détection d'erreurs.

CR3 contient l'adresse physique, alignée à 4 K-octets, de la première page.
La pagination fait partie du mode protégé.

Ainsi le 80386 offre quatre possibilités :
— un espace de 4G continu (*flat*)
— un espace segmenté
— un espace paginé
— un espace paginé et segmenté (page segmentée ou segment paginé).

Afin d'aider à la mise au point d'un système aussi complexe, nous disposons de registres de mise au point (*Debug*) et de test. Les premiers sont au nombre de 8 (DR0 à DR7) et autorisent 4 points d'arrêt (adresse sur 32 bits) contenus dans DR0 à DR3 ; DR4 et DR5 sont réservés, quant à DR6 et DR7 ils contiennent respectivement l'état (*status*) et le mot de contrôle des points d'arrêt.

Les registres de test, sont au nombre de 2 (TR6 et TR7) permettant de travailler avec des mémoires « caches » plus rapides que celles utilisées pour l'espace physique. Ce procédé autorise une grande vitesse d'exécution tout en réduisant le coût du plan mémoire.

Tous ces registres de 32 bits (CR0 à CR3, DR0 à DR7, TR6 et TR7) peuvent être lus et chargés par des instructions du type MOV.

Etat des registres après « reset »

Flags	XXXX0002	
CR0	XXXXXXX0	
EIP	0000FFF0	début en FFFFFFF0
CS	F000	avec un segment de 1 M-octet
DS, SS, ES, FS, GS	0000	de FFFF0000 à FFFFFFFF

(X : indéfini)

II — INTERRUPTIONS

Les interruptions sont celles du 286 complétées en mode protégé de l'interruption 14 concernant le mode paginé.

III — MODES D'ADRESSAGE

Nous disposons de tous les modes déjà vus, fournissant des adresses effectives sur 16 bits (espace adressable de 64 K-octets), complétés, bien sûr, par ceux du mode 32 bits (espace adressable de 4 G-octets) où interviennent, outre les registres EBX, EBP, ESI et EDI, les registres EAX, ECX et EDX. Nous pouvons passer du mode 16 bits au mode 32 bits, et réciproquement, grâce à un préfixe d'adresse — *Adress Size Prefix*.

Compte tenu de la taille des mots traités, 8, 16 ou 32 bits, l'adressage en table est facilité par l'emploi d'un facteur multiplicatif — *Scale factor* ou facteur d'échelle — qui multiplie le contenu d'un registre par 1, 2, 4 ou 8. Ainsi nous pouvons écrire :

```
MOV ECX, [EDX*8] [EAX]
```

qui charge le contenu de la case mémoire d'adresse effective (EAX) + (EDX)*8 par rapport à DS dans ECX.

IV — INSTRUCTIONS (*)

Comme nous l'avons dit, le 386 traite normalement des mots de 8 ou 16 bits (mode 8086) ou de 8 ou 32 bits (mode 386), mais on peut forcer le passage de 16 à 32 ou de 32 à 16 bits grâce à un préfixe — *Operand Size Prefix*. Ce préfixe et le préfixe d'adresse sont automatiquement ajoutés par l'assembleur selon l'écriture des instructions. Le mode 8086 autorise des données de 32 bits mais n'accepte que des adresses de 16 bits.
De plus, en mode protégé, un bit (D) de l'octet des règles d'accès des tables de description, indique la taille, par défaut, des mots et des adresses : 16 si D vaut 0 et 32 si D vaut 1.

Bit D	0	0	0	0	1	1	1	1
O.S.P.	Abs	Abs	Pré	Pré	Abs	Abs	Pré	Pré
A.S.P.	Abs	Pré	Abs	Pré	Abs	Pré	Abs	Pré
Mots de (bits)	16	16	32	32	32	32	16	16
Adresses de (bits)	16	32	16	32	32	16	32	16

O.S.P. Préfixe des mots — *Operand Size Prefix*
A.S.P. Préfixe des adresses — *Address Size Prefix*
Abs = Absent, Pré = Présent

Tableau 2. — Tailles des mots et des adresses.

IV-1 — Code machine

Le code machine des instructions comportera les octets suivants :

Préfixe	A.S.P.	O.S.P.	Segment	Code	ModRM	S.I.B.	Dep.	Donnée
0/1	0/1	0/1	0/1	1/2	0/1	0/1	0,1,2 ou 4	0,1,2 ou 4

où apparaissent les préfixes de segment, d'adresse et d'opérande.
Les chiffres indiquent le nombre d'octets représentatifs.
Nous voyons un nouvel octet : S.I.B. (*Scale Index Base*) intervenant dans le mode d'adressage 32 bits lorsque le programmeur utilise le facteur multiplicatif. Dans ce cas les bits de « mod » dans l'octet ModRM sont différents de 11 et les bits « r/m » sont à 100.
Cet octet contient le facteur multiplicatif, l'Index et la Base :

Scale	Index	Base

(*) **Note** : Les instructions que nous verrons provoquent en mode 8086 et réel une interruption de type 13 si l'espace mémoire dépasse 64 K.

ainsi dans l'exemple précédent

MOV ECX, [EDX*8] [EAX]

ECX est défini par les bits « reg » de l'octet ModRM (001), EDX par les bits *Index* (010), le facteur 8 par les bits *Scale* (11) et EAX par les bits *Base* (000) de l'octet S.I.B. Les tableaux 3 et 4 donnent les diverses possibilités offertes par ModRM et S.I.B.

IV-2 — Durée

Les durées des instructions sont pratiquement les mêmes pour le 386 que pour le 286, nous nous limiterons donc à donner ici les extensions des instructions du 286 et la description des nouvelles.

IV-3 — Extensions des instructions du 286 (*)

Les extensions sont de deux types, celles dues au préfixe de donnée, et celles dues à l'apparition de nouveaux registres.

IV-3-1 — Extensions dues au préfixe

Il s'agit d'instructions 16 bits étendues à 32 bits en mode 16 bits, grâce au préfixe de donnée, les codes machines étant identiques, à l'octet de préfixe près. Ainsi :

— **PUSHA** et **POPA** qui sauvent et restaurent les registres 16 bits sont étendues aux registres 32 bits en **PUSHAD** et **POPAD**.

— **PUSHF** et **POPF** qui concernent les flags sont étendues à EFLAGS en **PUSHFD** et **POPFD**.

mod	r/m	Adresse Effective	mod	r/m	Adresse Effective
	000	[EAX]		000	[EAX]+d32
	001	[ECX]		001	[ECX]+d32
	010	[EDX]		010	[EDX]+d32
00	011	[EBX]	10	011	[EBX]+d32
	100	S.I.B. est présent		100	S.I.B. est présent
	101	d32		101	[EBP]+d32
	110	[ESI]		110	[ESI]+d32
	111	[EDI]		111	[EDI]+d32

Tableau 3. — Mode d'adressage 32 bits sans S.I.B.

(*) **Note** : pour plus de précision on se reportera à la description de l'instruction 16 bits.

mod	r/m	Adresse Effective	mod	r/m	Registre
	000	[EAX]+d8		000	EAX/AX/AL
	001	[ECX]+d8		001	ECX/CX/CL
	010	[EDX]+d8		010	EDX/DX/DL
01	011	[EBX]+d8	11	011	EBX/BX/BL
	100	S.I.B. est présent		100	ESP/SP/AH
	101	[EBP]+d8		101	EBP/BP/CH
	110	[ESI]+d8		110	ESI/SI/DH
	111	[EDI]+d8		111	EDI/DI/BH

d8 : donnée de 8 bits étendue, signée à 32 bits
d32 : donnée de 32 bits
Le segment est SS pour une adresse effective où interviennent EBP ou ESP, DS pour les autres modes.

Tableau 3. (suite) — Mode d'Adressage 32 bits sans S.I.B.

ss	index	base
00 x1	000 EAX	000 EAX
01 x2	001 ECX	001 ECX
10 x4	010 EDX	010 EDX
11 x8	011 EBX	011 EBX
	100 — —	100 ESP
	101 EBP	101 selon mod
	110 ESI	110 ESI
	111 EDI	111 EDI

mod	base = 101
00	d32
01	[EPB]+d8
10	[EBP]+d32

Tableau 4. — Mode d'Adressage 32 bits et l'octet S.I.B.

Les instructions de conversion 8 en 16 bits **CBW** et 16 en 32 bits **CWD** sont respectivement étendues à :

- **CWDE** qui met dans EAX le nombre, signé, contenu dans AX (extension, signée, de 16 en 32 bits).
- **CDQ** qui met dans la concaténation (EDX:EAX) le contenu de EAX (extension, signée, de 32 à 64 bits).

Remarque : on parlera de mots (W — 16 bits), de double-mots (DW — 32 bits) et de quadruple-mots (QW — 64 bits).

— **JCXZ,** saut si le contenu de CX est nul, est étendu à **JECXZ** ; le nombre de boucles autorisées passe ainsi de 64 K (65536) à 4 G (4 294 967 296 !).

IV-3-2 — Extensions dues aux nouveaux registres

— Extension des préfixes

L'existence de deux registres de segments supplémentaires FS et GS conduit à deux nouveaux préfixes de segments — *Override Prefix :*

FS : codé	0110 0100	soit 64 H
GS : codé	0110 0101	soit 65 H

Il ne faut oublier les préfixes de donnée et d'adresse :

— Préfixe d'adresse — *Adress Size Prefix* (passage de 16 à 32 bits et vice-versa)

codé	0110 0111	soit 67 H

— Préfixe de donnée — *Operand Size Prefix* (passage de 16 à 32 bits et vice-versa)

codé	0110 0110	soit 66 H

— Extensions de MOV

MOV a été étendu aux registres de contrôle (CR_0 à DR_7) et aux registres de test. (TR6 et TR7).

1. Chargement des registres de contrôle depuis un registre

Code machine

00001111	00100010	00 eee reg	

avec

eee	Registre de contrôle	Durée
000	CR0	10
010	CR2	4
011	CR3	5

Exemple :

MOV CR0, EAX codée 0F 22 00

2. Chargement des registres de mise au point depuis un registre

Code machine

00001111	00100011	11 eee reg	

avec

eee	Registre de mise au point	Durée
000	DR0	22
001	DR1	22
010	DR2	22
011	DR3	22
110	DR6	16
111	DR7	16

Exemple :

MOV DR6, EAX codée 0F 23 F0

3. Chargement des registres de Test depuis un registre

Code machine

00001111	00100110	11 eee reg	

avec

eee	Registre de Test	Durée
110	TR6	12
111	TR7	12

Exemple :

MOV TR6, EAX codée 0F 26 F0

4. Chargement d'un registre avec le contenu d'un :

4-1. Registre de Contrôle

Code machine

0000 1111	0010 0000	00 eee reg	

Durée : 6

Exemple :

MOV EAX, CR3 codée 0F 20 18

4-2. Registre de mise au point

Code machine

0000 1111	0010 0001	11 eee reg	

Durée : 22 pour DR0 à DR3 et 14 pour DR6 et DR7

Exemple :

MOV EAX, DR2 codée 0F 21 D0

4-3. Registre de Test

Code machine

0000 1111	0010 0100	11 eee reg	

Durée : 12

Exemple :

MOV EAX, TR7 codée 0F 24 F8

Remarque 1 : le premier octet est 0F comme pour toutes les nouvelles instructions.

Remarque 2 : il n'existe pas de MOV pour FS et GS qui n'interviennent, hors les préfixes, que dans les instructions suivantes.

Extension de POP et PUSH

— POP sauvegarde d'un registre de segment en pile

Code machine

0000 1111	10 s reg 001		

Durée : 7

avec

s reg	Segment
100 101	FS GS

Exemple :

POP FS codée 0F A1

— PUSH restauration d'un registre de segment depuis la pile

Code machine

0000 1111	10 s reg 000		

Durée : 2

Exemple :

PUSH GS codée 0F A8

Extension de LDS, LES

— LFS chargement du registre de segment FS et d'un registre depuis la mémoire

Code machine

0000 1111	1011 0100	mod reg r/m	

Durée : 7 mod ≠11

— LGS chargement du registre de segment GS et d'un registre depuis la mémoire

Code machine

0000 1111	1011 0101	mod reg r/m	

Durée : 7 mod ≠11

— LSS chargement du Stack Segment et d'un registre depuis la mémoire

Code machine

0000 1111	1011 0010	mod reg r/m	

mod ≠11

Durée : 7

Très utile pour créer une pile temporaire en initialisant à la fois SS (registre de segment de pile) et SP ou BP, par exemple par

LSS BP, PILE_LOCALE

IV-3-3 — Extension d'espace adressable

Cette extension concerne les sauts conditionnés, relatifs, dont l'espace est porté à 16 bits (+32 767 à −32 768) ou 32 bits (+2 147 483 647 à −2 147 483 648) et les sauts inconditionnés.

— Sauts conditionnés

Code machine

0000 1111	1000 cccc	décalage 2 ou 4 octets

avec

Mnémoniques	Condition	cccc
JO	Overflow	0000
JNO	Pas d'Overflow	0001
JB / JNAE	Inférieur	0010
JNB / JAE	Supérieur ou égal	0011
JE / JZ	Egal	0100
JNE / JNZ	Différent	0101
JBE / JNA	Inférieur ou égal	0110
JNBE / JA	Supérieur	0111
JS	Négatif	1000
JNS	Positif	1001
JP /JPE	Parité Paire	1010
JNP / JPO	Parité Impaire	1011
JL / JNGE	Plus petit	1100
JNL / JGE	Plus grand ou égal	1101
JLE / JNG	Plus petit ou égal	1110
JNLE / JG	Plus grand	1111

Durée : Pas de saut : 3
Saut effectif 7+m où m indique le nombre d'octets de l'instruction qui suit (il faut vider la file d'attente).

— Sauts inconditionnés

Les sauts inconditionnés (JMP) voient leur espace adressable augmenté. Ainsi le saut « proche indirect » (ou intra-segment) voit son espace mémoire porté à 32 bits, par exemple :

JMP EAX

En ce qui concerne les sauts extra-segment, où la valeur du contenu de CS est modifiée, seul l'offset peut être de 32 bits, que ce soit par chargement immédiat ou via des cases mémoire.

Dans le premier cas nous aurons une adresse définie par 6 octets, 2 pour le contenu de CS et 4 pour celui de EIP.

V — NOUVELLES INSTRUCTIONS

Nous examinerons les nouvelles instructions par ordre alphabétique des mnémoniques.

BSF Recherche de Bit de droite à gauche

Cette intruction permet de trouver le premier bit non nul d'un mot, d'un double-mot, du contenu d'un registre ou de cases mémoire. La position de ce bit, repérée à partir du bit de **poids faible** — *Bit Scan Forward* — est chargée dans un registre de même longueur que le mot analysé. Le flag ZF est mis à zéro en cas de succès (un bit égal à un), sinon le flag ZF est mis à un et le contenu du registre destination indéfini.

Code machine

0000 1111	1011 1100	mod reg r/m	

Durée : 10+3n où n est le nombre de bits testés pour trouver un bit non nul

Flags : Affectés ZF
Indéfinis

Exemple :

BSF EDX, ESI codée 0F BC D6

Mettra le flag ZF à 0, et chargera EDX avec 00000008 si le contenu de ESI est :

xxxx xxxx	xxxx xxxx	xxxx xxx1	0000 0000	
31	23	15	7	0

les bits marqués x peuvent être indifféremment 1 ou 0, puisque la recherche s'arrête au premier bit égal à un rencontré.

BSR Recherche de Bit de gauche à droite

Cette instruction permet de trouver le premier bit non nul d'un mot, d'un double-mot, du contenu d'un registre ou de cases mémoire. La position de ce bit, repérée en partant du bit de poids fort — *Bit Scan Reverse* — est chargée dans un registre de même longueur que le mot analysé. Le flag ZF est mis à zéro en cas de succès (un bit égal à un), sinon le flag est mis à un et le contenu du registre destination, indéfini.

Code machine

0000 1111	1011 1101	mod reg r/m	

Durée : 10 + 3n où n est le nombre de bits testés pour trouver un bit non nul

Flags : Affectés ZF
Indéfinis

Exemple :

BSR EDX,ESI codée 0F BD D6

Mettra le flag ZF à 0, et chargera EDX avec 00000008 si le contenu de ESI est :

0000 0000	0000 0000	0000 0001	xxxx xxxx
31	23	15	7 0

les bits marqués x peuvent être indifféremment 1 ou 0, puisque la recherche s'arrête au premier bit égal à un rencontré.

BT Test de Bit

Cette instruction met dans le carry (CF) la valeur du bit situé dans le premier opérande (*base*) — registre ou case mémoire — à la position indiquée par le deuxième opérande (*offset*), ce dernier pouvant être un registre ou une donnée, l'instruction possède deux codes machine. Dans le cas de cases mémoire, l'offset est signé, et le repérage a lieu à partir du bit de poids faible pour un offset positif, du bit de poids fort pour un offset négatif ; l'octet concerné est celui dont l'adresse est donnée.

1. Registre ou mémoire, donnée

Code machine

0000 1111	1011 1010	mod 100 r/m	donnée

la donnée est « limitée » à 4 bits (16) pour un registre de 16 bits, 5 bits (32) pour un registre de 32 bits ; elle est signée et limitée 16 et 32 bits dans le cas de cases mémoire.
Dans ce dernier cas elle peut être nulle (absente).

Durée : registre 3
mémoire 6

Flags : Affectés CF
Indéfinis

Exemple :

BT EAX, 21 codée 0F BA E0 15

met dans CF le bit 21 de EAX

Remarque : pour le repérage des bits dans les registres et les cases mémoire, voir l'instruction IBTS.

2. Registre ou mémoire, registre

Code machine

0000 1111	1010 0011	mod reg r/m	

Durée : registre 3
mémoire 12

Flags : Affectés CF
Indéfinis

Exemples :

1 — BT EAX, EBX codée 0F A3 D8

est équivalent à BT EAX, 21 si le contenu de EBX est 21

2 — BT [BX], AX codée 0F A3 07

BTC Test de Bit et Complément

Cette instruction opère comme BT, mais après avoir mis le bit désiré dans CF, elle le complémente.
L'offset du bit peut être le contenu d'un registre ou une donnée dont la valeur ne peut dépasser la taille du registre testé.

1. Registre ou mémoire, donnée

Code machine

0000 1111	1011 1011	mod 111 r/m	donnée

Durée : registre 6
mémoire 8

Flags : Affectés CF
Indéfinis

Exemple :

BTC EAX, 21 codée 0F BA F8 15

met le bit 21 de EAX dans le flag CF et le complémente

2. Registre ou mémoire, registre

Code machine

0000 1111	1011 1011	mod reg r/m	

Durée : registre 6
mémoire 13

Flags : Affectés CF
Indéfinis

Exemples :

1 — BTC EAX, EBX codée 0F BB D8

réalise la même opération que BTC EAX, 21 si le contenu de EBX est 21

2 — BTC [BX], AX codée 0F BB 07

BTR Test de Bit et mise à zéro
Cette instruction opère comme BT, mais après avoir mis le bit désiré dans CF, elle le met à zéro (*reset*).
L'offset du bit peut être une donnée dont la valeur ne doit pas dépasser la taille du registre testé, ou le contenu d'un registre.

1. Registre ou mémoire, donnée

Code machine

0000 1111	1011 1010	mod 110 r/m	donnée

Durée : registre 6
mémoire 13

Flags : Affectés CF
Indéfinis

Exemple :

BTR EAX, 21 codée 0F BA F0 15

met dans CF le bit 21 de EAX, puis le met à zéro

2. Registre ou mémoire, registre

Code machine

0000 1111	1011 0011	mod reg r/m	

Durée : registre 6
mémoire 13

Flags : Affectés CF
Indéfinis

Exemples :

1 — BTR EAX, EBX codée 0F B3 D8

réalise la même opération que BTR EAX, 21 si le contenu de EBX est 21

2 — BTR [BX], AX codée 0F B3 07

BTS Test de Bit et mise à un

Cette instruction opère comme BT, mais après avoir mis le bit désiré dans CF, elle le met à un (*set*).
L'offset du bit peut être une donnée dont la valeur ne doit pas dépasser la taille du registre testé, ou le contenu d'un registre.

1. Registre ou mémoire, donnée

Code machine

0000 1111	1011 1010	mod 101 r/m	donnée

Durée : registre 6
mémoire 8

Flags : Affectés CF
Indéfinis

Exemple :

BTS EAX, 21 codée 0F BA E8 15

met dans CF le bit 21 de EAX, puis le met à un

2. Registre ou mémoire, registre

Code machine

0000 1111	1010 1011	mod reg r/m	

Durée : registre 6
mémoire 13

Flags : Affectés CF
Indéfinis

Exemples :

1 — BTS EAX, EBX codée 0F AB D8

réalise la même opération que BTS EAX, 21 si le contenu de EBX est 21

2 — BTS [BX], AX codée 0F AB 07

IBTS Insertion d'une Suite de Bits

Cette instruction range une suite de bits dont la longueur est définie par le contenu de CL, justifiée à droite, contenue dans un registre, dans un mot (16 bits) ou un double-mot (32 bits) — registre ou cases mémoire. La position du premier bit stocké (*offset*) est le contenu, signé, de AX ou EAX ; si la destination est un registre, l'offset doit être positif. Dans tous les cas il ne doit pas y avoir de débordement, sinon le résultat est indéfini. L'ordre des termes est : destination, offset, longueur, source.

Code machine

0000 1111	1010 0111	mod reg r/m	

Durée : registre 12
mémoire 19

Flags : Affectés aucun
Indéfinis

Exemples :

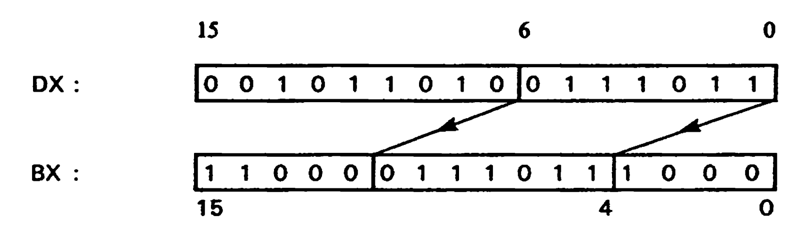
1 — IBTS BX, AX, CL, DX codée 0F A7 D3

(BX ↑ r/m ; DX ↑ reg)

Si CL contient 07, AX 04 et si les contenus de DX et BX sont respectivement :

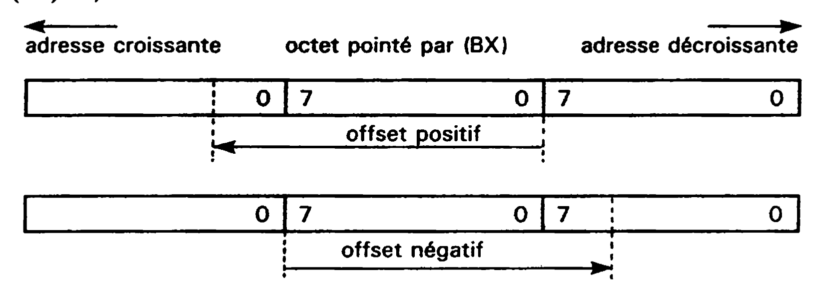
DX : 0010110100111011
BX : 1100010011111000

nous aurons l'opération suivante :



2 — IBTS DWORD PTR [BX], EAX, CL, EDX codée 0F A7 17

Si les contenus de CL, EAX sont identiques à l'exemple ci-dessus, la modification concernera les bits des octets pointés par (BX) et (BX) + 1 — offset positif — par contre si le contenu de EAX était négatif, la modification intéresserait les octets pointés par (BX) et (BX) − 1, selon le schéma ci-dessous :



0 et 7 sont les numéros des bits extrêmes des octets

Remarque : il s'agit de l'écriture d'un ensemble de bits, on peut utiliser cette instruction pour charger les bits de poids forts des registres 32 bits (EAX...).

IMUL Multiplication Signée

Le 386 reprend, bien sûr, les multiplications de 286, étendues éventuellement à 32 bits, mais il ajoute une multiplication, signée entre un registre, de 16 ou 32 bits, à la fois multiplicande et résultat, et un registre ou un mot mémoire de 16 ou 32 bits.
Les flags CF et OF sont mis à zéro si le résultat est correct, du point de vue format, sinon ils sont mis à un.

Code machine

0000 1111	1010 1111	mod reg r/m	

Durée : registre × registre 9 à 38
registre × mémoire 12 à 41

Flags : Affectés OF et CF mis à 0 ou 1
Indéfinis SF, ZF, AF, PF

Exemple :

IMUL EDX, EBX codée 0F AF D3

Multiplie le contenu, signé de EBX par le contenu, signé, de EDX. Le résultat, limité à 32 bits, sera stocké, signé, dans EDX.

MOVSX Transfert avec extension signée

Cette instruction est équivalente à un « MOV », mais la destination, un registre, est de taille supérieure à la source, registre ou cases mémoire. Les bits manquants seront égaux au bit de poids fort de la quantité à « étendre », conservant ainsi son signe. On peut passer de 8 à 16 bits, de 8 à 32 et de 16 à 32 bits.

Code machine

0000 1111	1011 111w	mod reg r/m	

w = 0 la source est de 8 bits, la destination de 16 ou 32 bits précisée, éventuellement par un préfixe de donnée
w = 1 la source est de 16 bits

Durée : registre 3
mémoire 6

Flags : Affectés aucun
Indéfinis

Exemple :

MOVSX EAX, AL codée 0F BE C0

si le bit de poids fort du contenu de AL est à 1 (nombre négatif) les 24 bits de poids fort de EAX seront égaux à 1

MOVZX Transfert avec extension nulle

Cette instruction est équivalente à un « MOV », mais la destination, un registre, est de taille supérieure à la source, registre ou cases mémoire. Les bits manquants seront égaux à zéro. On peut passer de 8 à 16 bits, de 8 à 32 et de 16 à 32 bits.

Code machine

0000 1111	1011 011w	mod reg r/m	

w = 0 la source est de 8 bits, la destination de 16 ou 32 bits, précisée éventuellement par un préfixe de donnée
w = 1 la source est de 16 bits

Durée : registre 3
mémoire 6

Flags : Affectés Aucun
Indéfinis

Exemple :

MOVZX EAX, AL codée 0F B6 C0

les 24 bits de poids fort de EAX seront égaux à zéro

SETcc Mise à un des bits d'un octet sous condition

Cette instruction permet la mise à un de tous les bits d'un octet (*SET Byte*) si une condition est remplie, sinon cet octet sera mis à zéro. Les conditions sont celles des sauts conditionnés.

Code machine

0000 1111	1001 cccc	mod 000 r/m	

avec

Mnémoniques	Condition	cccc
SETO	Overflow	0000
SETNO	Pas d'Overflow	0001
SETB/SETNAE	Inférieur	0010
SETNB/SETAE	Supérieur ou égal	0011
SETE/SETZ	Egal	0100
SETNE/SETNZ	Différent	0101
SETBE/SETNA	Inférieur ou égal	0110
SETNBE/SETA	Supérieur	0111
SETS	Négatif	1000
SETNS	Positif	1001
SETP/SETPE	Parité Paire	1010
SETNP/SETPO	Parité Impaire	1011
SETL/SETNGE	Plus petit	1100
SETNL/SETGE	Plus grand ou égal	1101
SETLE/SETNG	Plus petit ou égal	1110
SETNLE/SETG	Plus grand	1111

Durée : registre 4
mémoire 5

Flags : Affectés aucun
Indéfinis

Exemples :

1 — SETL AL codée 0F 9C C0

met (AL) à FF si SF⊕OF vaut 1 (voir les sauts conditionnés de 8086) et à 00 dans le cas contraire

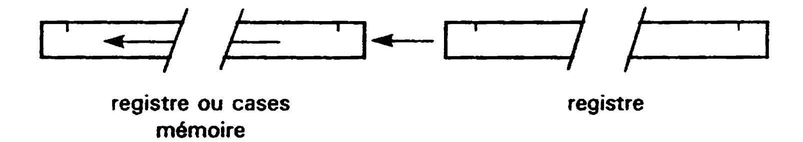
2 — SETS BYTE PTR [BX] codée 0F 98 07

met à FF l'octet pointé par (BX) par rapport à (DS) si le flag SF est à 1, sinon cet octet sera mis à 00

Remarque : les flags testés sont ceux affectés par la **dernière** opération pouvant les modifier.

SHLD Décalage à Gauche en Double Précision

Cette instruction opère un décalage (*Shift*) à gauche (*Left*) sur 16 ou 32 bits du contenu d'un registre ou de cases mémoire, d'un nombre de bits égal à la valeur indiquée ou au contenu de CL (limités à 32), en injectant, à partir du bit de poids faible, les bits d'un registre qui ne sera pas modifié, selon le schéma :



registre ou cases mémoire registre

1. Valeur indiquée par une donnée

Code machine

0000 1111	1010 0100	mod reg r/m	donnée

Durée : registre 3
mémoire 7

Flags : Affectés SF, ZF, PF, en accord avec le résultat final, CF est égal au dernier bit « sorti » du registre
Indéfinis AF, OF

Exemple :

```
          r/m   reg
           ↓     ↓
SHLD EBX, EDX, 4            codée 0F A4 D3 04
```

éliminera les 4 bits de poids fort de (EBX) les « remplaçant » par les 4 bits de poids fort de (EDX) qui ne sera pas modifié.
Ainsi si le contenu de EBX est : FE32F023
celui de EDX : 874FEC12 nous aurons après exécution de l'instruction ci-dessus :

(EBX) = E32F028 et (EDX) = 874FEC12

avec SF = 1, ZF = 0, PF = 1 et CF = 0

2. Valeur indiquée par CL

Code machine

0000 1111	1010 0101	mod reg r/m	

Durée : registre 3
mémoire 7

Flags : Affectés SF, ZF, PF, CF
Indéfinis AF, OF

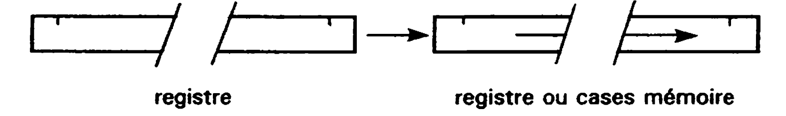
Exemple :

```
SHLD EBX, EDX, CL            codée 0F A5 D3
```

est équivalente à SHLD EBX, EDX, 4 si (CL) est égal à 4, mais le contenu de CL n'est pas détruit...

SHRD Décalage à Droite en Double Précision

Cette instruction opère un décalage (*Shift*) à droite (*Right*) sur 16 ou 32 bits du contenu d'un registre ou de cases mémoire, d'un nombre de bits égal à la valeur indiquée ou au contenu de CL (limités à 32), en injectant, à partir du bit de poids fort, les bits d'un registre, qui n'est pas modifié, selon le schéma :



1. Valeur indiquée par une donnée

Code machine

0000 1111	1010 1100	mod reg r/m	donnée

Durée : registre 3
mémoire 7

Flags :	
Affectés	SF, ZF, PF, CF
Indéfinis	AF, OF

Exemple :

```
     r/m  reg
      ↓    ↓
SHRD EBX, EDX, 4            codée 0F AC D3 04
```

éliminera les 4 bits de poids faible de (EBX) les remplaçant par les 4 bits de poids faible de (EDX) qui ne sera pas modifié.
Ainsi si le contenu de EBX est : FE32F023
celui de EDX : 874FEC12 nous aurons après exécution de l'instruction ci-dessus :

(EBX) = 2EF32F02 et (EDX) = 874FEC12

avec SF = 0, ZF = 0, PF = 1 et CF = 0

2. Valeur indiquée par CL

Code machine

0000 1111	1010 1101	mod reg r/m	

Durée : registre 3
mémoire 7

Flags :	
Affectés	SF, ZF, PF, CF
Indéfinis	AF, OF

Exemple :

```
SHRD EBX, EDX, CL           codée 0F AB D3
```

est équivalente à SHRD EBX, EDX, 4 si (CL) est égal à 4, mais le contenu de CL n'est pas détruit...

XBTS Extraction d'une Suite de Bits

Cette instruction extrait une suite de bits dont la longueur est définie par le contenu de CL, justifiée à droite, d'un mot ou d'un double-mot — registre ou cases mémoire — et la met dans un registre. La position du premier bit extrait (*offset*) est le contenu, signé, de AX ou EAX ; si la source est un registre, l'offset doit être positif.
Dans tous les cas il ne doit pas y avoir de débordement, sinon le résultat est indéfini.

Code machine

0000 1111	1010 0110	mod reg r/m	

Durée : registre 6
mémoire 13

Flags : Affectés aucun
Indéfinis

Exemples :

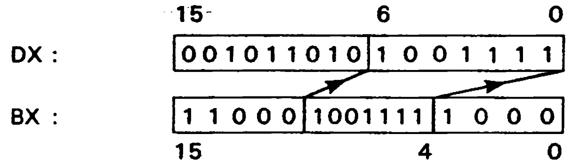
reg r/m

1 — XBTS DX, BX, AX, CL codée 0F A6 D3

si CL contient 07, AX 04 et si les contenus de DX et BX sont :

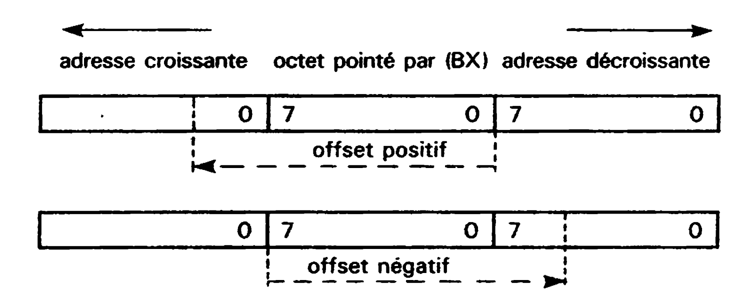
DX : 0010110100111011
BX : 1100010011111000

nous aurons l'opération suivante :



2 — XBTS DWORD PTR [BX], EAX, CL, EDX codée 0F A7 17

Si les contenus de CL, EAX sont identiques à l'exemple ci-dessus, la modification concernera les bits des octets pointés par (BX) et (BX) + 1 — offset positif — par contre si le contenu de EAX était négatif, la modification intéresserait les octets pointés par (BX) et (BX) − 1, selon le schéma ci-dessous :



0 et 7 sont les numéros des bits extrêmes des octets

Remarque : il s'agit d'une lecture d'un ensemble de bits, on peut utiliser cette instruction pour lire les bits de poids fort des registres 32 bits (EAX...).

INTERFAÇAGE

Le 8086 peut travailler selon deux modes :

— un mode dit « minimum ». Le processeur est seul à gérer son espace mémoire. Ce mode peut être « bufférisé » ou non selon le nombre de composants connectés au bus. En effet, la charge maximale autorisée est de 100 pF (100 picofarads) pour garantir les caractéristiques temporelles du processeur (temps de montée et descente des signaux essentiellement) avec un courant limité à 2 mA. Or chaque composant présente une capacité d'entrée, exprimée en picofarad, et les capacités mises en parallèle s'ajoutent. Le système maximal, non bufférisé, est de trois périphériques (20 pF par circuit) et quatre mémoires (5 à 12 pF par circuit). Il faut donc bufffériser en ajoutant un amplificateur (8286 ou 8287) autorisant un débit de 32 mA pour une charge maximale de 300 pF. Il faudra bien sûr tenir compte des temps de transit dans le choix des types de mémoire (temps d'accès), certaines configurations exigeant un temps d'attente (*Wait state*).

— un mode maximum : le processeur est associé à un coprocesseur arithmétique (8087), au processeur entrée/sortie (8089) ou contrôle une carte dans un système multiprocesseur.

Ces deux modes sont obtenus en reliant l'une des pattes (MN/$\overline{MX}$) du circuit à la masse ou au +5 volts. Dans le mode maximum, les signaux de contrôle — lecture ($\overline{RD}$) ou écriture ($\overline{WR}$) — sont gérés par un circuit annexe (8288) qui différencie mémoires et périphériques adressés via les instructions IN/OUT.

Les brochages du 8086 et du 8088 avec les deux possibilités, le mode maximum étant indiqué entre parenthèses, sont donnés figure 1.

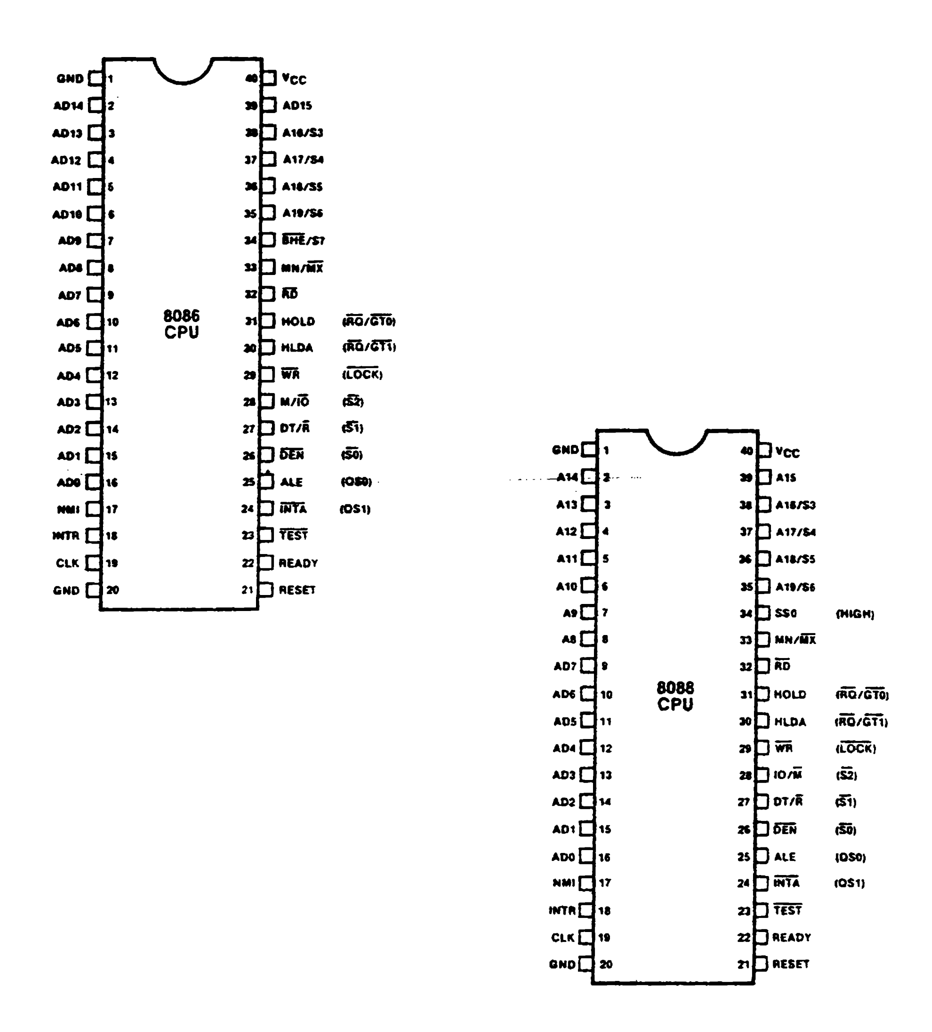
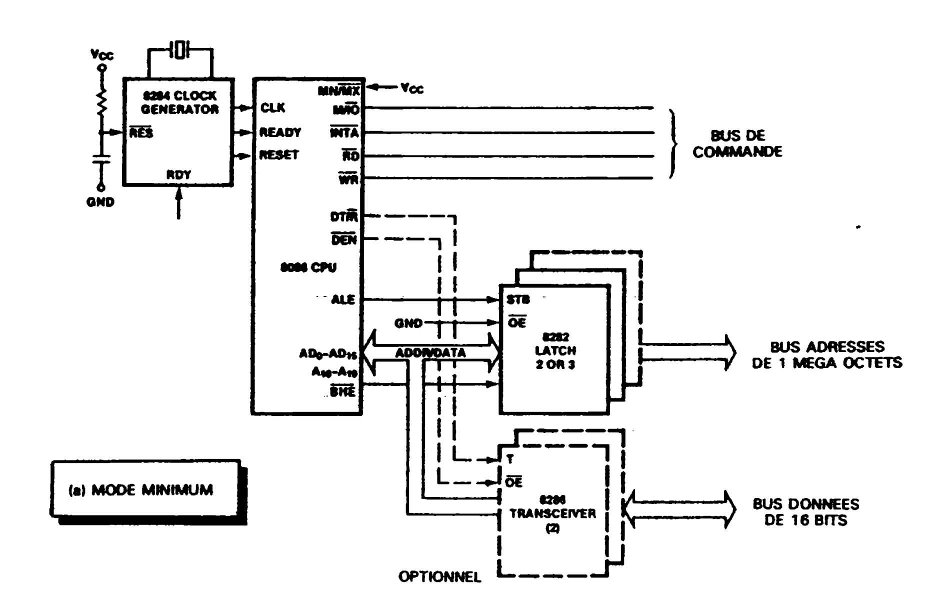


Fig. 1. — Brochage du 8086 et du 8088. Le mode maximum est indiqué entre parenthèses.

La figure 2 montre les connexions à réaliser et les composants de base pour les deux modes ; vous remarquerez la présence, en mode minimum, des buffers 8286, optionnels, permettant d'augmenter la charge du bus des données.



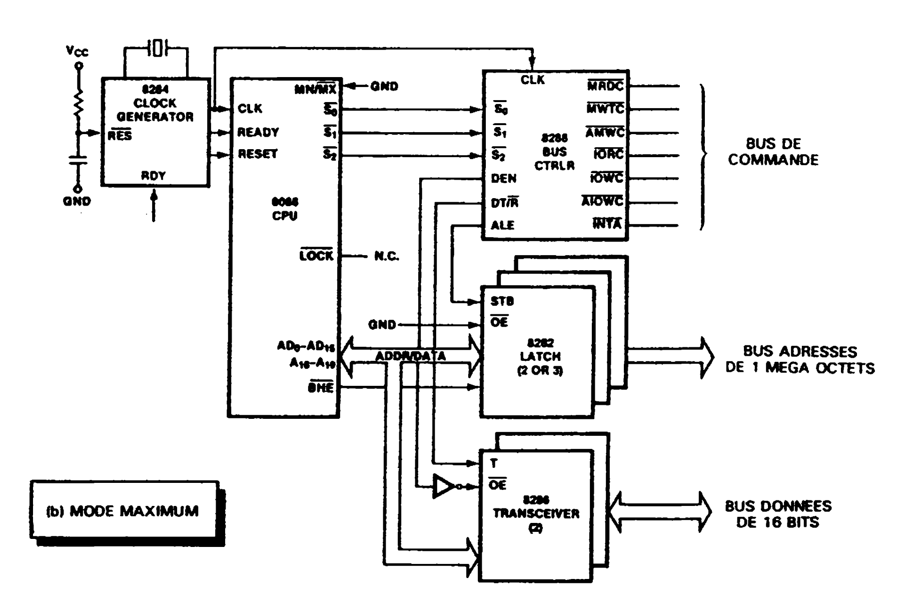
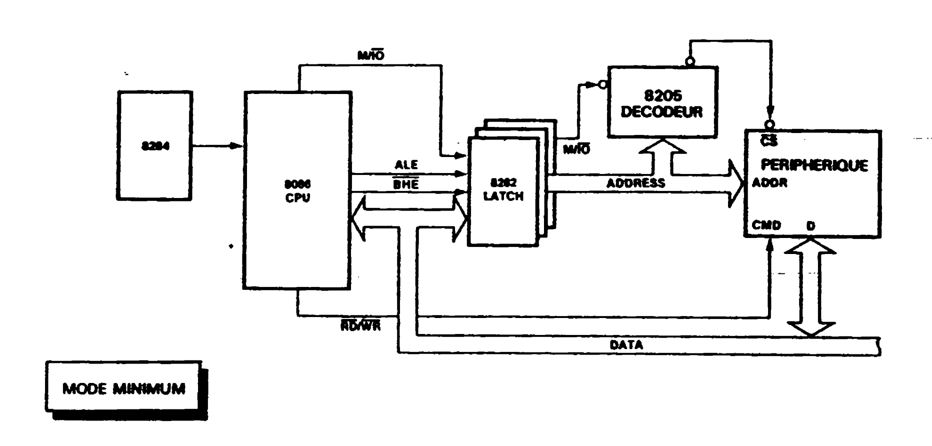


Fig. 2. — Configurations de base en mode minimum (a) et maximum (b).

Les périphériques (ports parallèles, USART...) câblés en mode « I/O » (entrée/sortie) permettent l'usage des instructions IN et OUT, et occupent les 64 premiers K-octets de l'espace mémoire — 00000 à 0FFFF — La figure 3 montre les composants nécessaires, et les connexions à réaliser en mode minimum, « bufférisé » ou non ; le mode maximum étant décrit figure 4 dans les mêmes conditions.



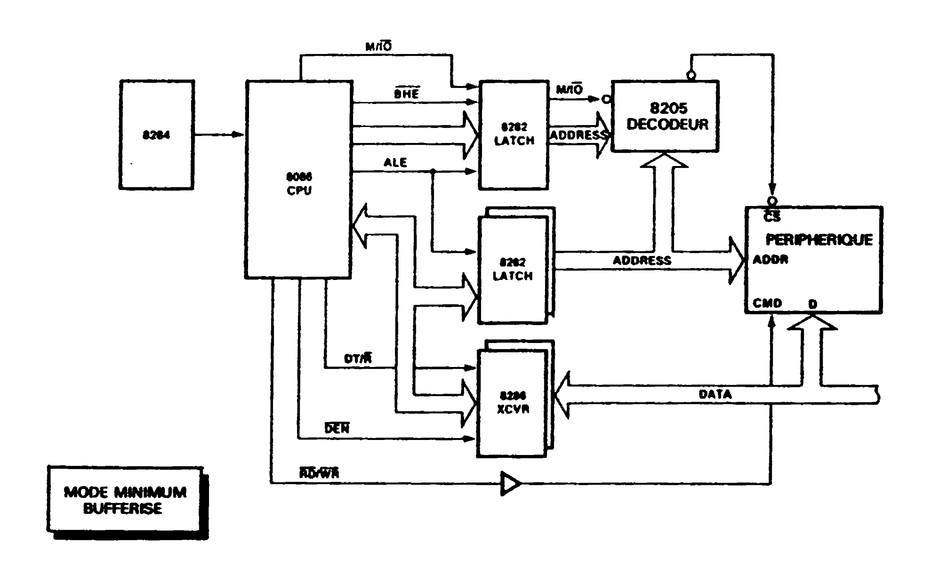


Fig. 3. — Périphérique en mode minimum.

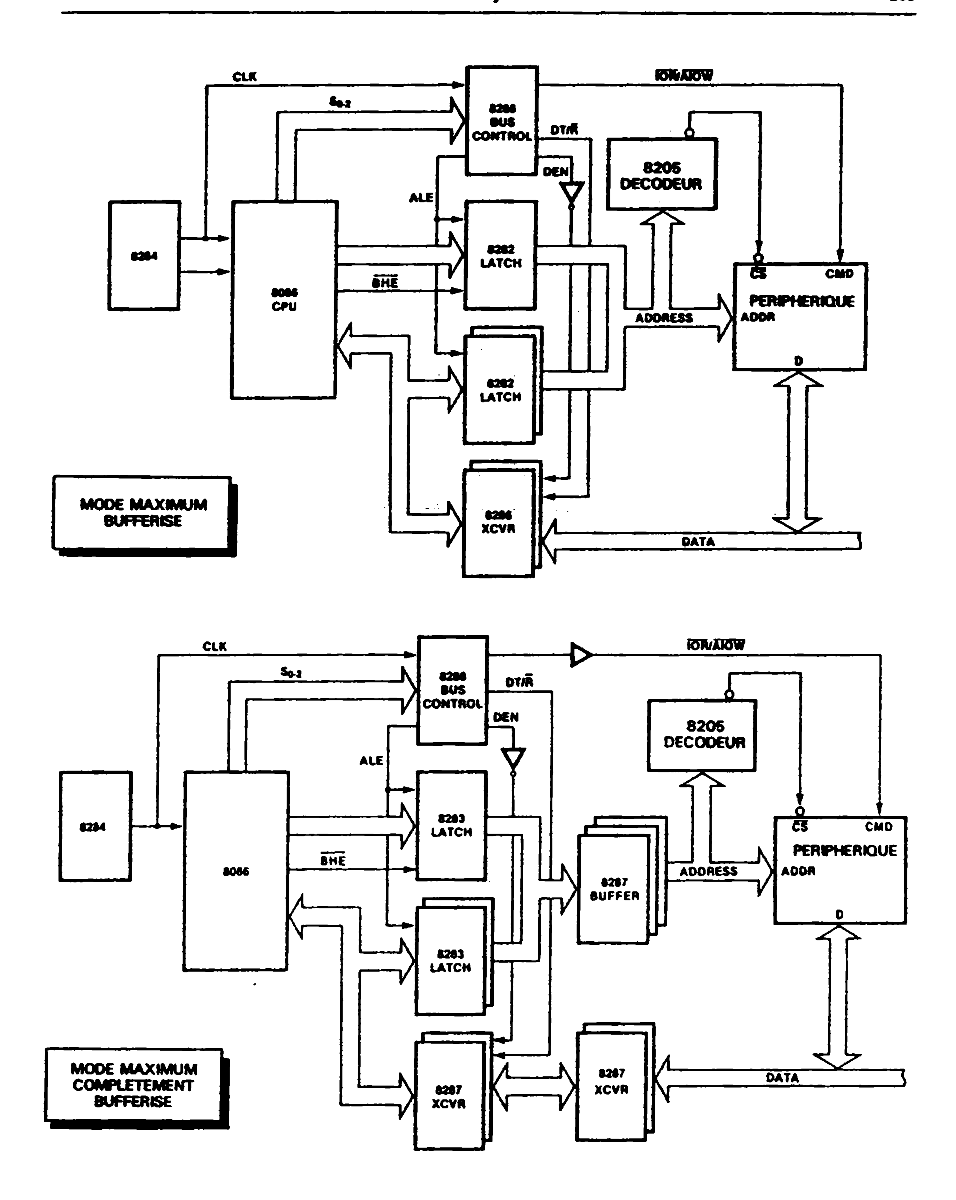


Fig. 4. — Périphérique en mode maximum.

Compte tenu des retards apportés par les circuits d'interface — latch, buffer, décodeurs d'adresses — il peut être nécessaire de prévoir un temps d'attente pour certains périphériques « lents ». Le tableau 1 indique les « wait-states » pour un 8086 travaillant à 5 MHz.

Configuration				
Mode minimum			Mode maximum	
Non bufférisé		Bufférisé	Bufférisé	Totalement bufférisé
2716-1	—	—	—	—
2716-2	—	1W	1W	1W
2732	1W	1W	1W	1W
2332	—	—	—	—
2364	—	—	—	—
8251A	—	1W	—	—
8253-5	—	1W	—	—
8255A-5	—	1W	—	—
8257-5	—	1W	—	—
8259A	—	—	—	—
8271	—	1W	—	—
8273	—	1W	—	—
8275	—	1W	—	—
8279-5	—	1W	—	—
8041A*	—	1W	—	—
8741A	—	1W	—	—
8291	—	—	—	—
(*) Y compris les périphériques dérivés du 8041A (8292, 8294, 8295).				

Tableau 1. — Wait-states (W) à prévoir pour quelques mémoires et périphériques (8086 à 5 MHz).

A titre d'exemple, nous donnons le schéma de la carte d'apprentissage SDK 86 (figure 5) comportant :

— 8 K-octets d'EPROM (4 boîtiers 2716) contenant le moniteur, et situés dans le « haut » de l'espace mémoire en raison des valeurs attribuées aux contenus de CS et IP à la mise sous tension ou après un Reset (FFFF et 0000 pour commencer en FFFF0) ;

— 2 K de RAM de 00000 à 007FF pouvant être étendus à 4 K ;

— un USART (8251) assurant la liaison série d'adresses FFF0 et FFF2 ;

— un contrôleur de clavier et d'affichage 7-segments (8279) d'adresses FFE8 et FFEA ;

— 2 boîtiers de ports parallèles (8255) d'adresses FFF8 à FFFF, offrant 3 ports de 16 bits ou 6 ports de 8 bits.

Les périphériques sont câblés I/O.

La sélection des EPROMs ne peut être assurée à l'aide de décodeurs du type 8205 ou 74LS138, puisque les bits de poids fort des adresses sont à 1 après un Reset, il faut donc utiliser soit des PROM soit des PAL (*Programmable Array Logic* : réseaux logiques programmables). Ici, le décodage est assuré par des PROM de 1 K*4 du type 3625 dont les seules données stockées sont E (1110), D (1101), B (1011) et 7 (0111) aux emplacements adéquats — la sélection ayant lieu pour 0. Les figures 6 à 9 donnent le détail des connexions réalisées autour du 8086 (6), des EPROMs (7), des RAMs (8) et de l'USART (9) qui peut travailler à 75, 110, 150, 300, 600, 1200, 2400 et 4800 bauds, et dont les entrées-sorties sont adaptables au mode « boucle de courant » (TTY) ou au mode RS232 (CRT).

Le 8086 de cette carte est câblé en mode minimum (MN/$\overline{\text{MX}}$) à 5 volts, si l'on désire expérimenter le coprocesseur arithmétique 8087, il sera nécessaire de convertir notre système en mode maximum, à l'aide d'une carte qui se connectera à la place du processeur et comportant, outre le 8087, quelques circuits dont le contrôleur 8288. Le schéma permettant le passage d'un mode à l'autre est donné par la figure 10 et les connexions à réaliser autour du 8087, par la figure 11.

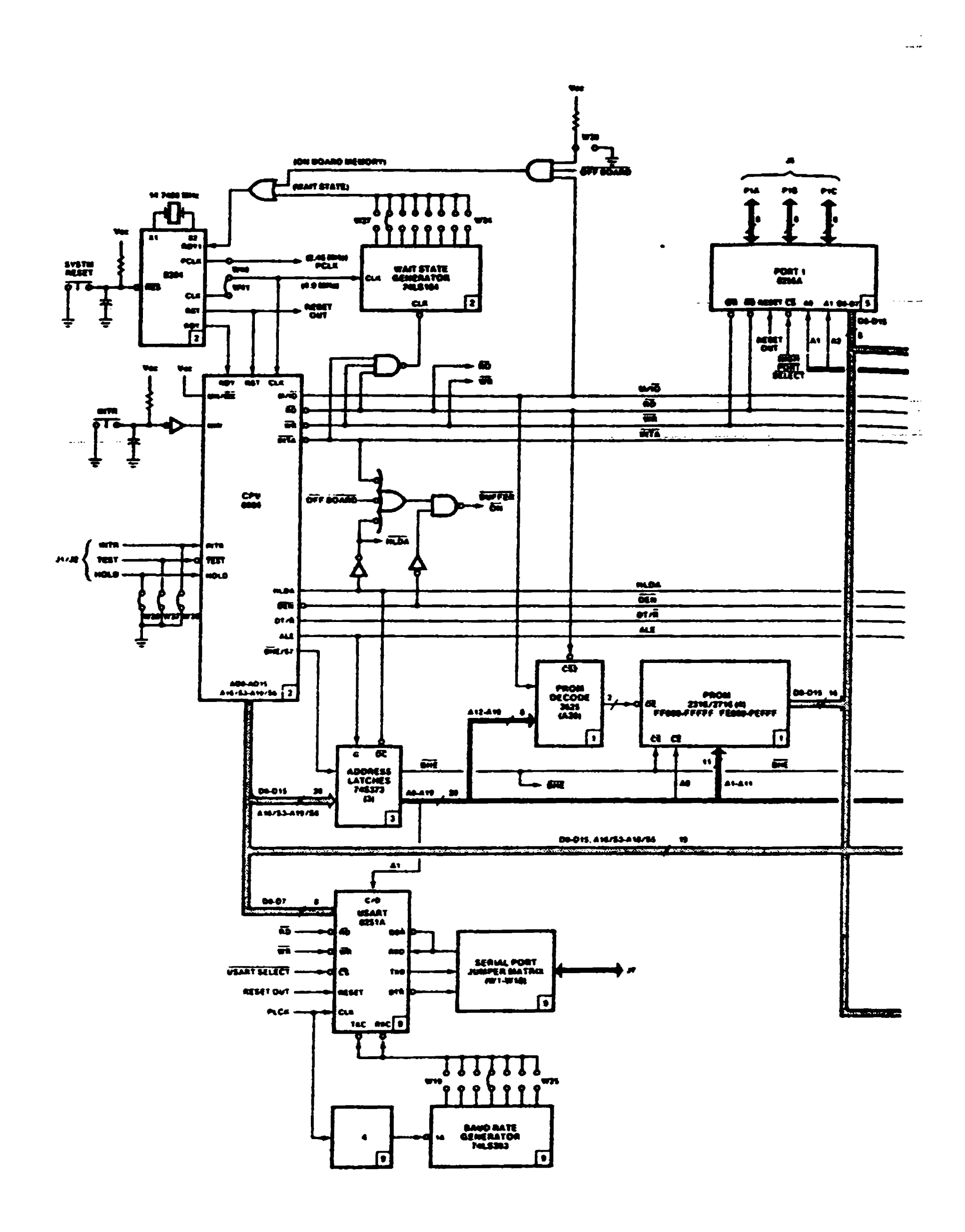


Fig. 5. — Schéma de la carte SDK 86.

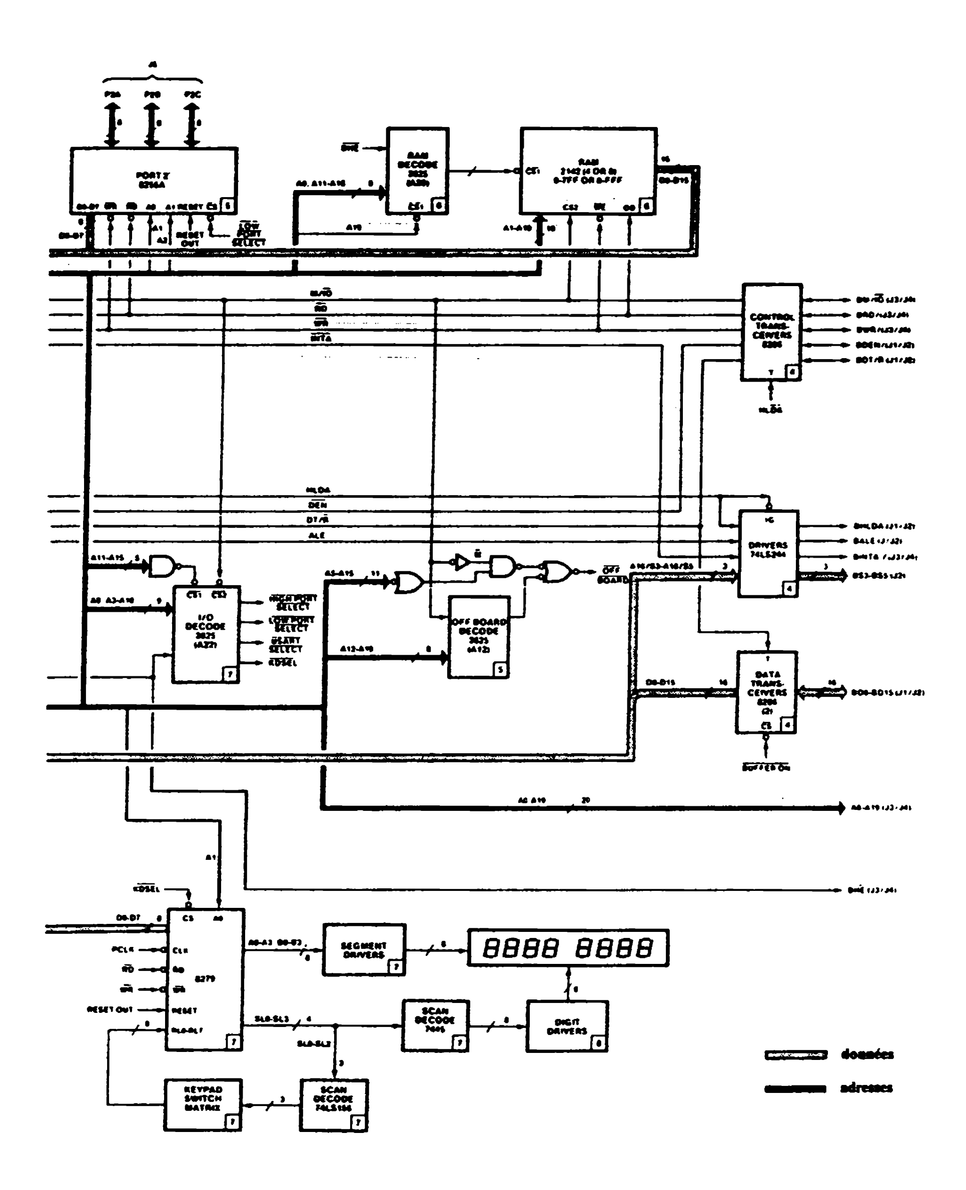


Fig. 5. — Schéma de la carte SDK 86.

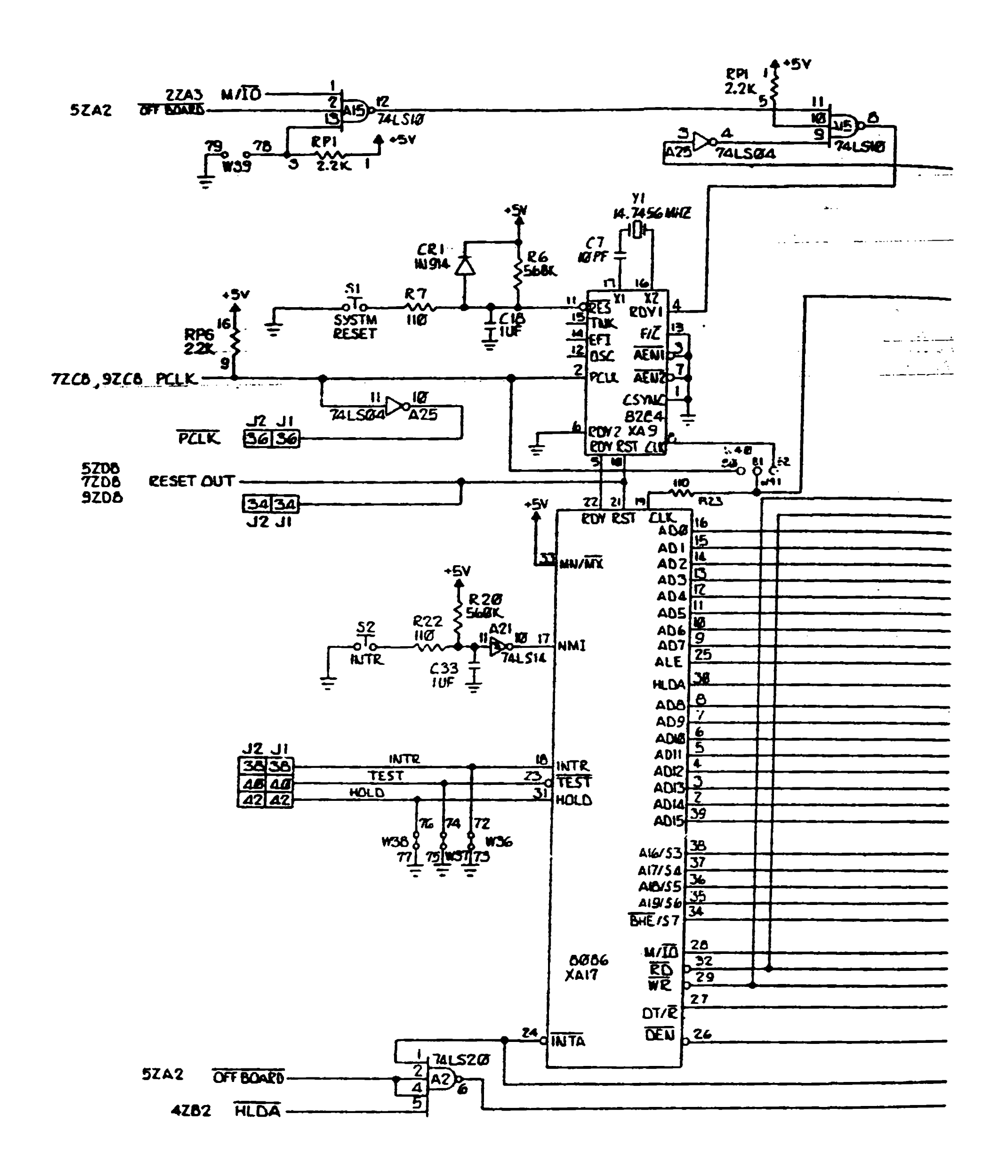


Fig. 6. — Câblage du 8086.

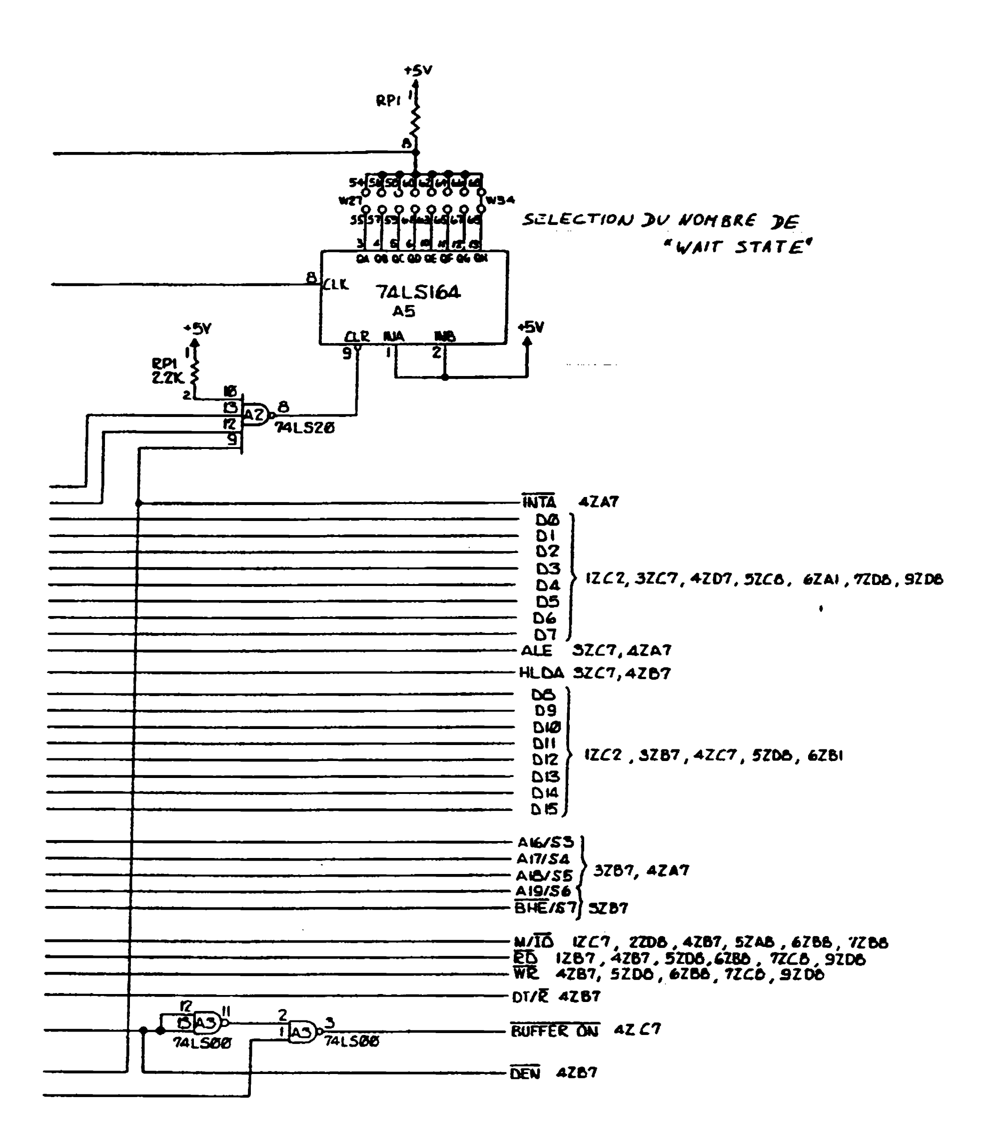
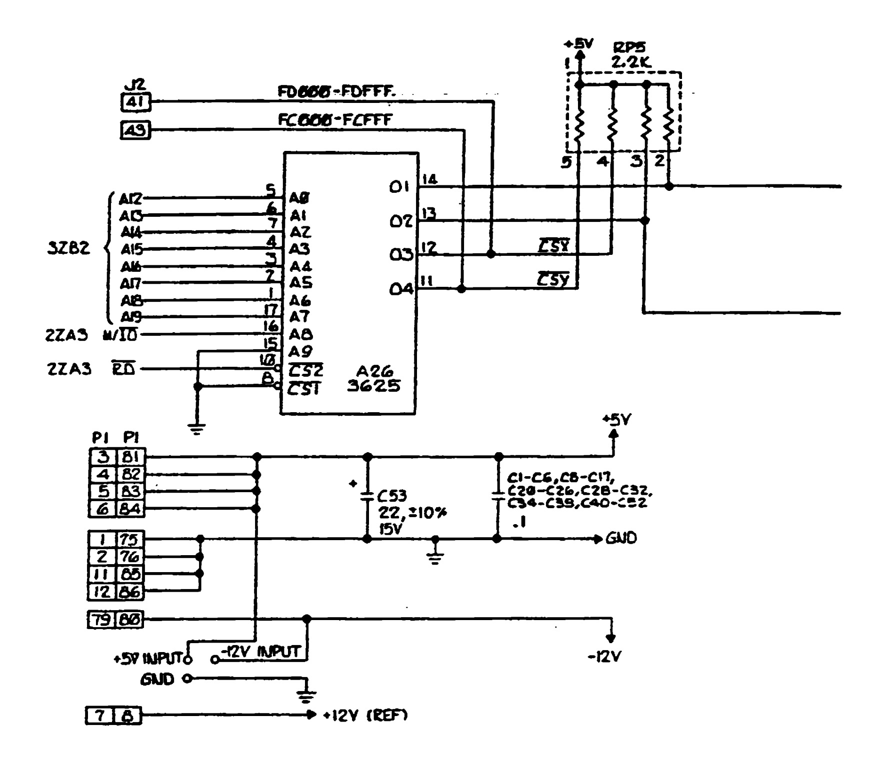


Fig. 6. — Câblage du 8086.



TOUTES DIODES 1N914B
TOUS TRANSISTORS Q2T 2905
TOUTES LES CAPACITES SONT EN μF +80 −20 %, 50 V
TOUTES LES RESISTANCES SONT EN OHMS ±5 % 1/4 WATT

Fig. 7. — Câblage des EPROM.

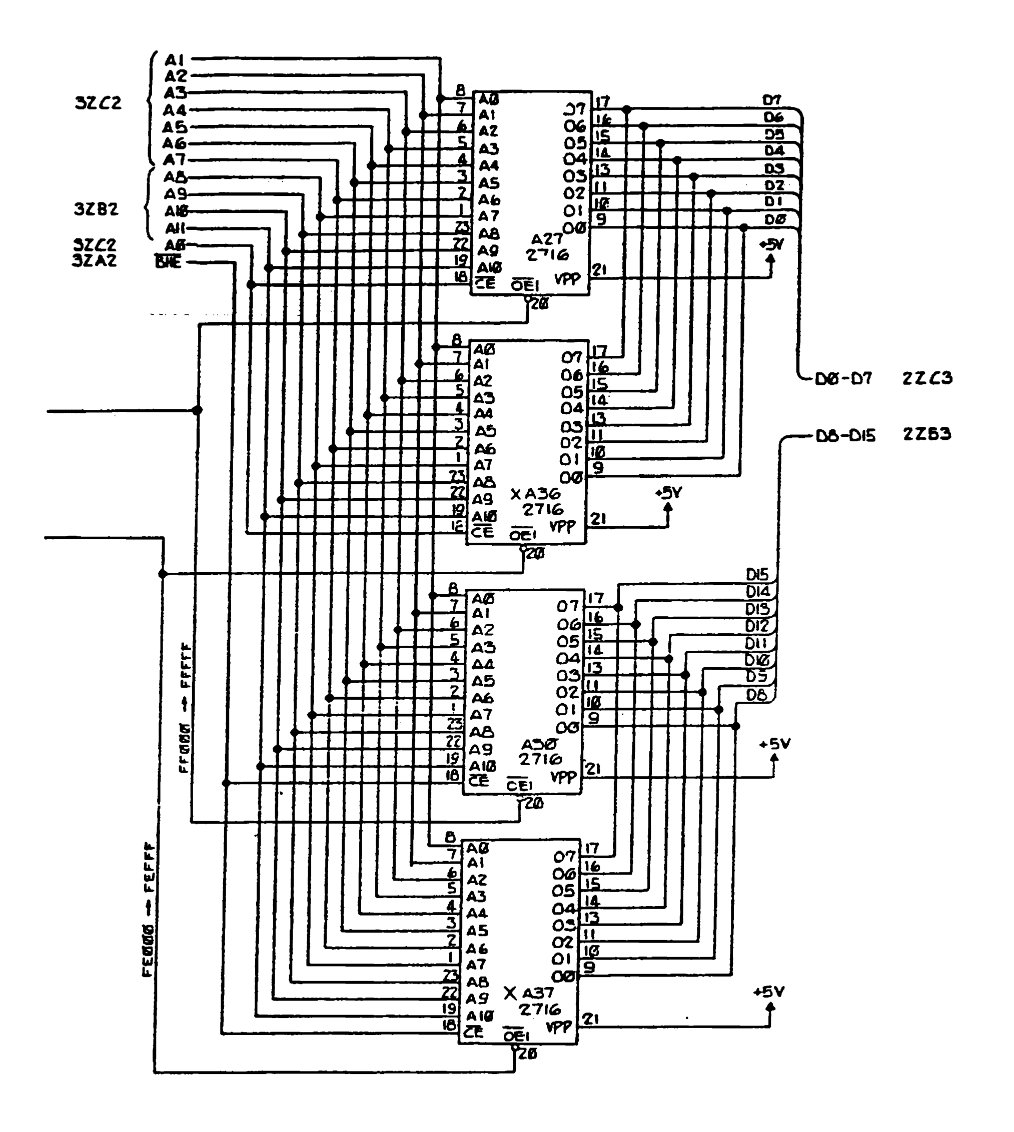


Fig. 7. — Câblage des EPROM.

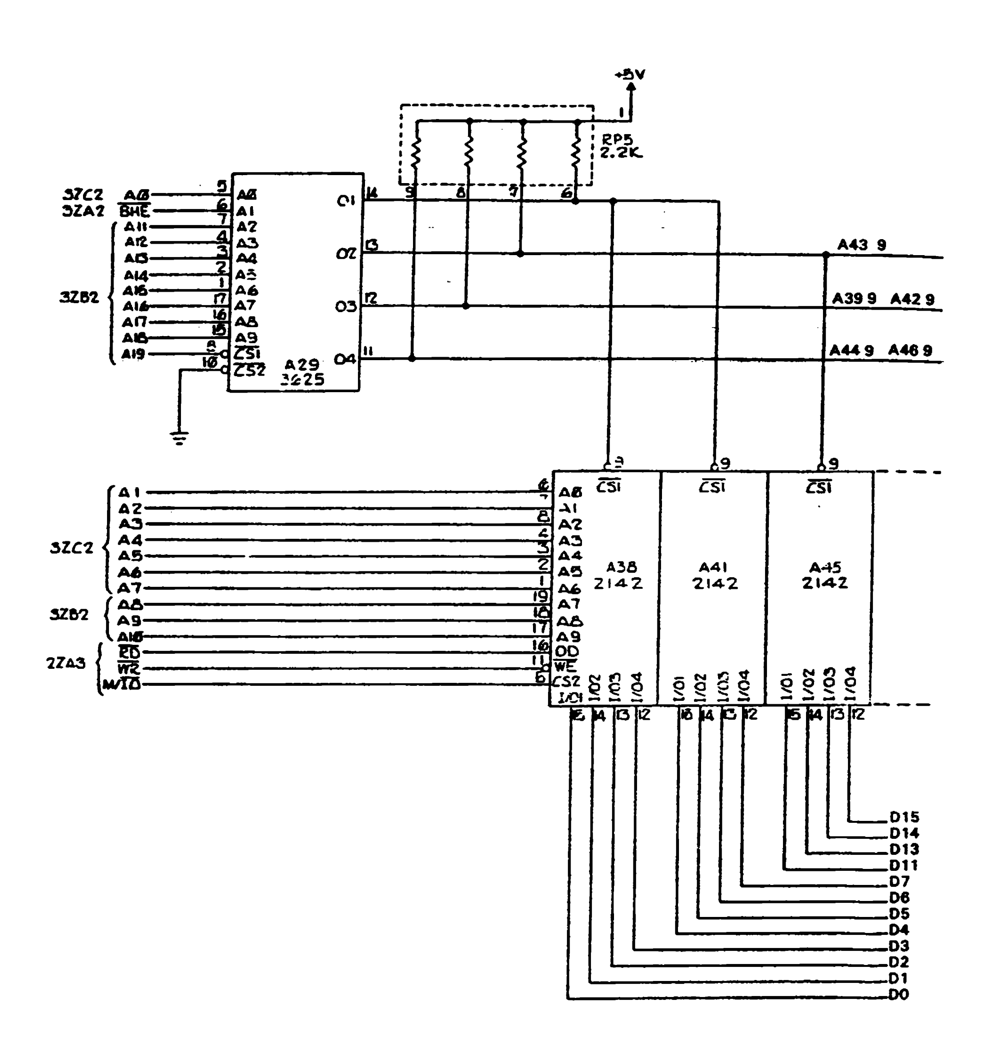


Fig. 8. — Câblage des RAM.

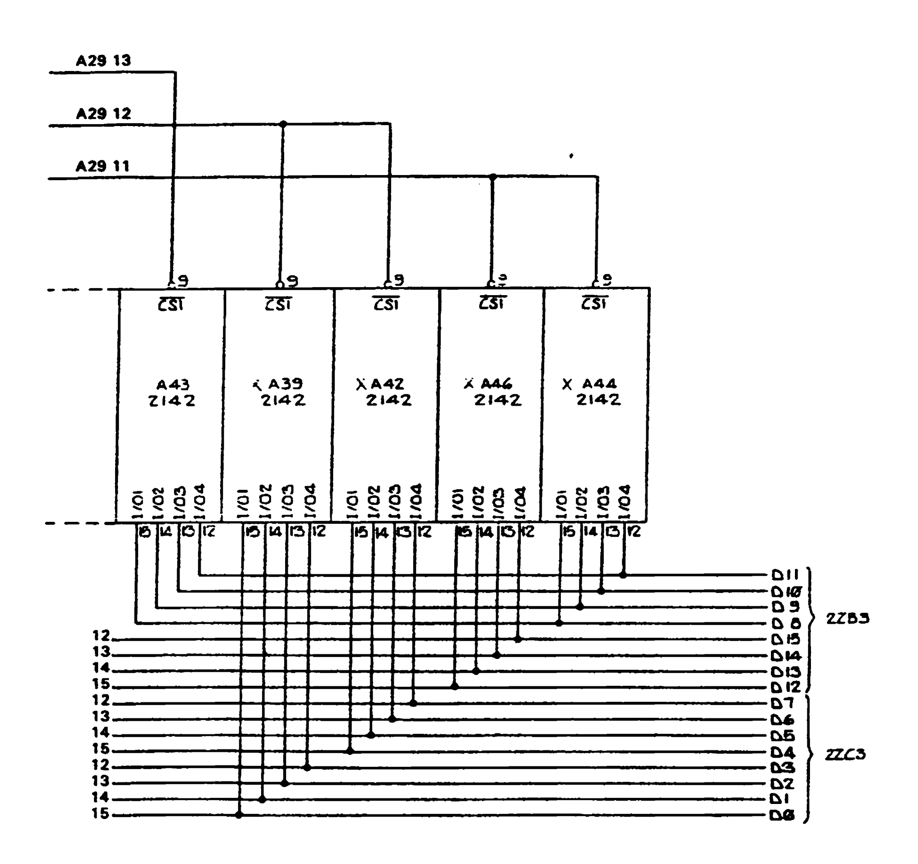


Fig. 8. — Câblage des RAM.

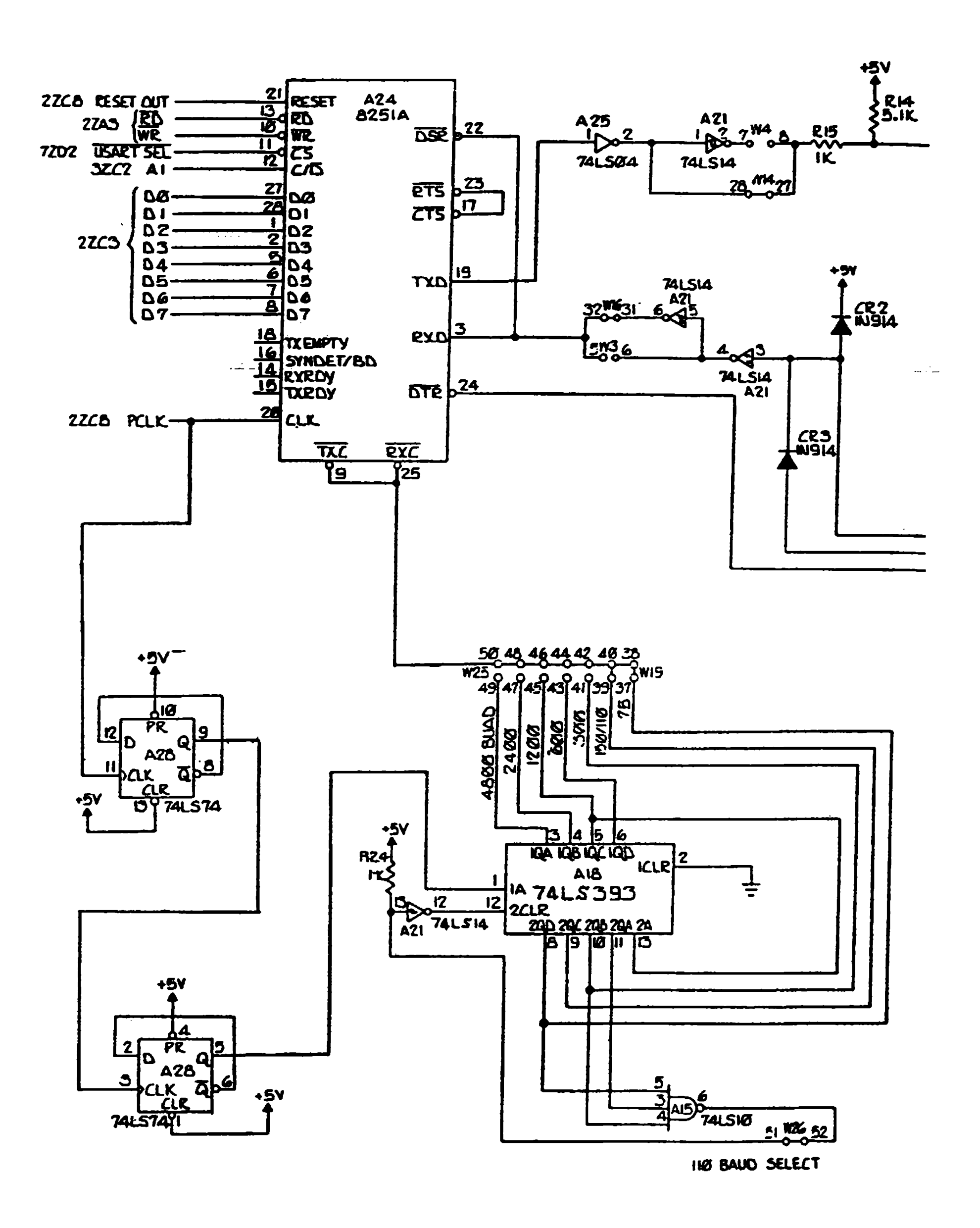
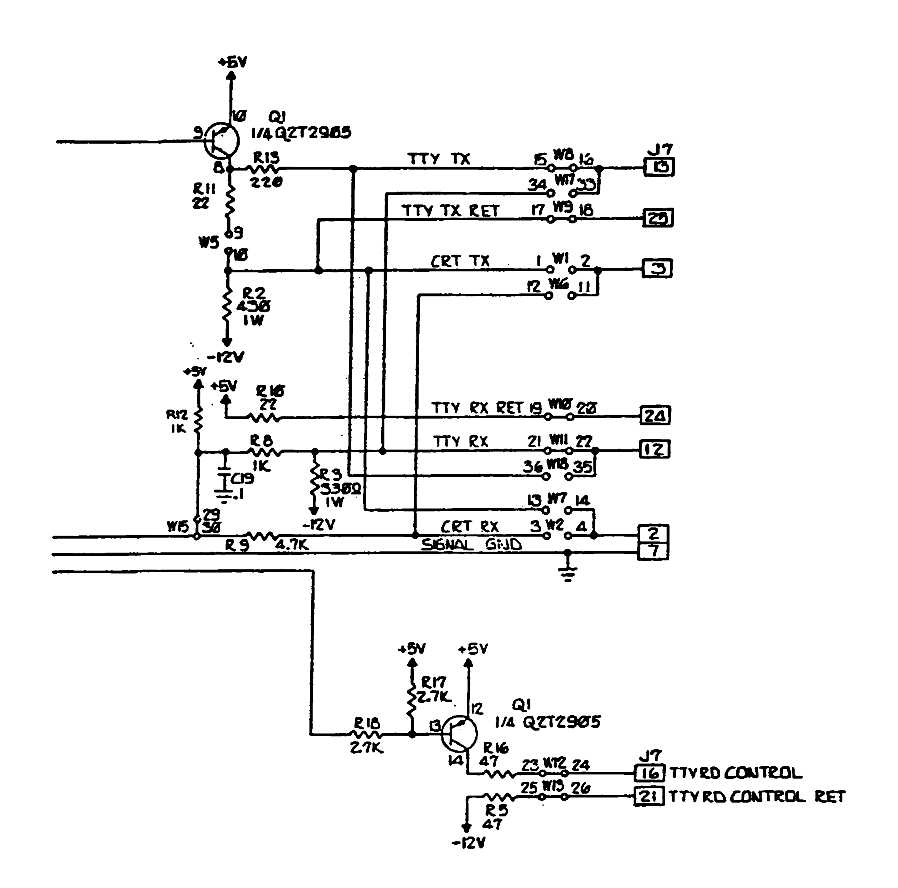


Fig. 9. — Câblage de l'USART.



SERIAL INTERFACE JUMPER TABLE			
STAND ALONE		MDS SLAVE	
CRT	TTY	CRT PORT	TTY PORT
1-2	15-16	5-6	27-28
3-4	17-18	7-8	29-3Ø
5-6	19-2Ø	9-1Ø	31-32
7-8	21-22	11-12	33-34
9-1Ø	23-24	13-14	35-36
	25-26		
	27-28		
	29-3Ø		
	31-32		

Fig. 9. — Câblage de l'USART.

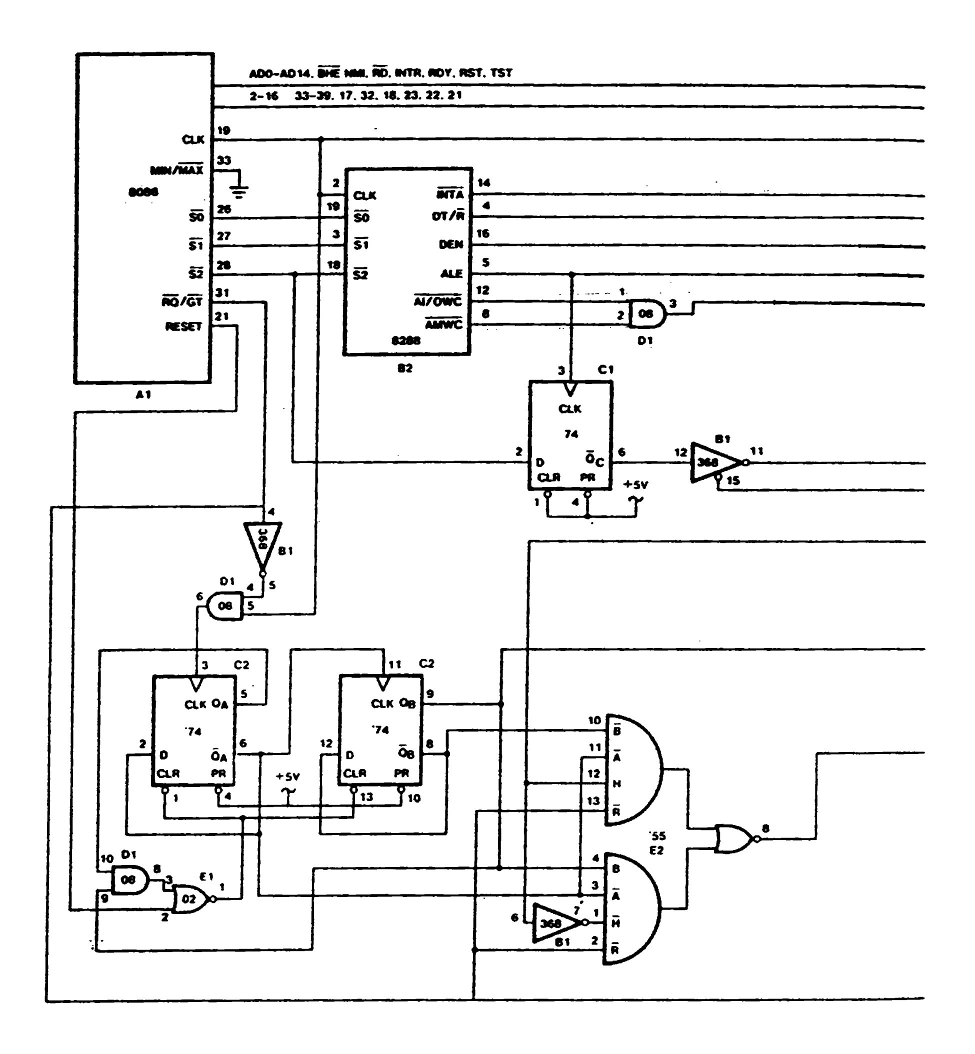


Fig. 10. — Schéma de conversion du mode minimum au mode maximum (remarquez la présence du 8288).

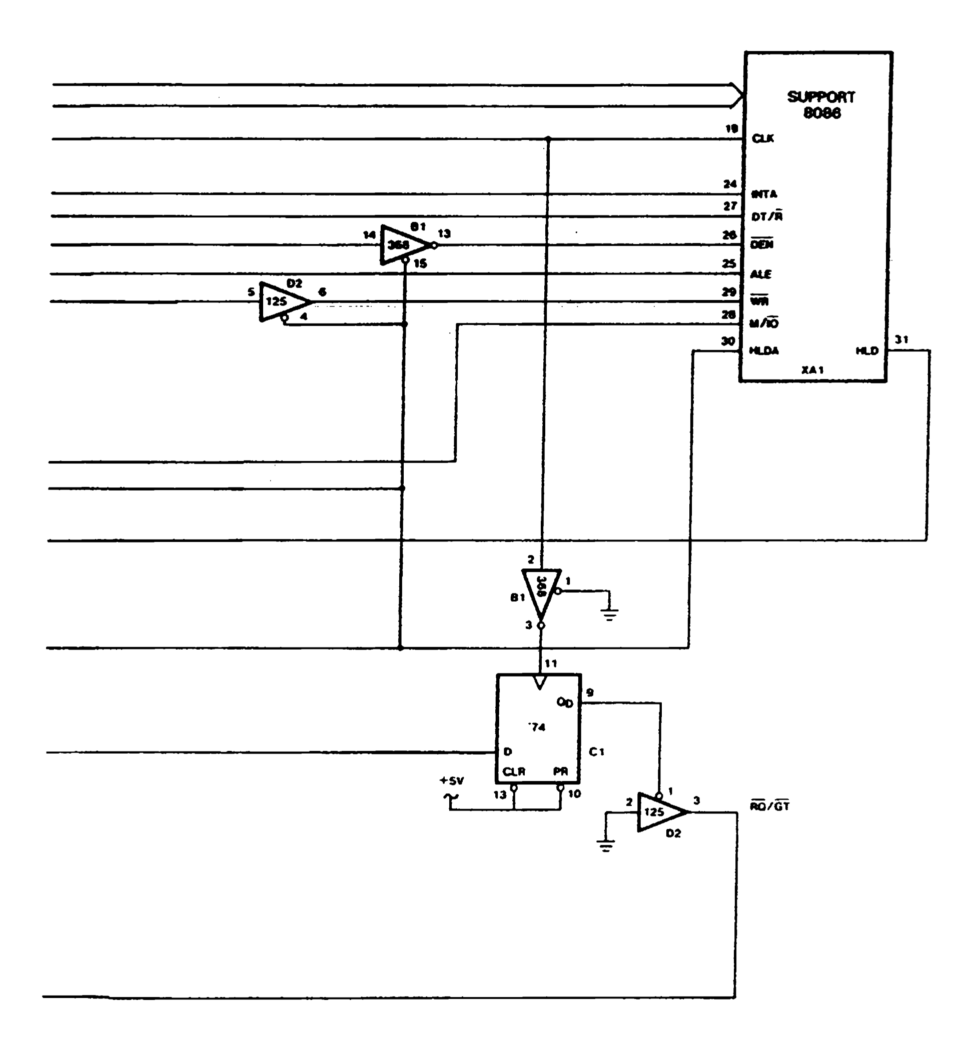
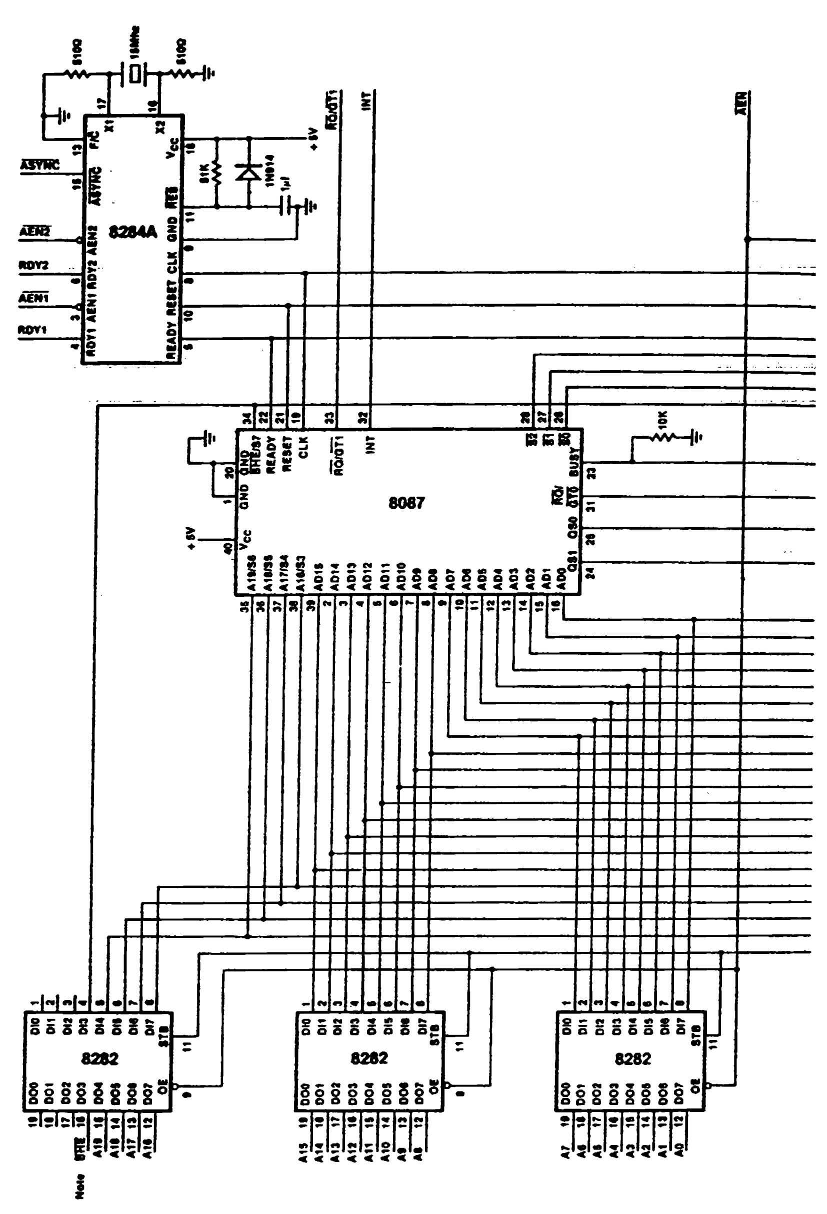
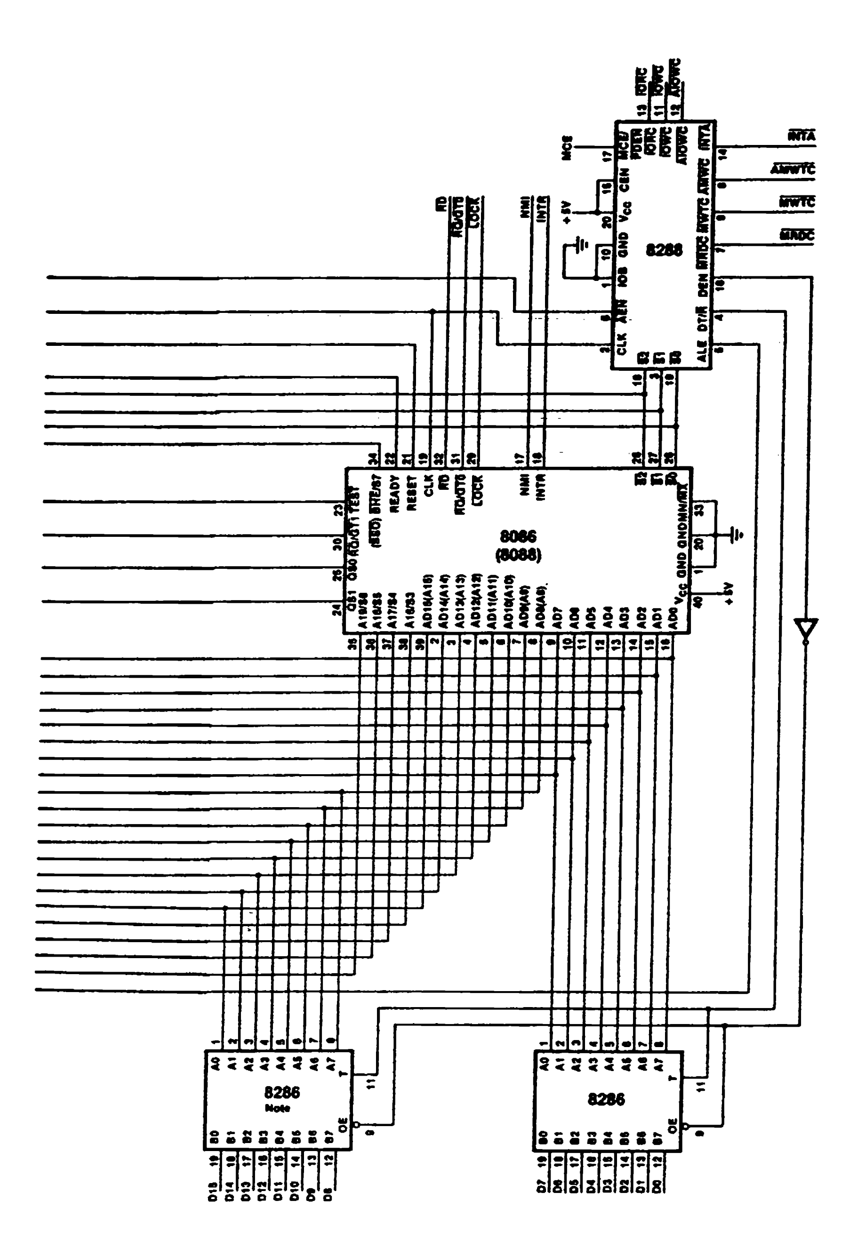


Fig. 10. — Schéma de conversion du mode minimum au mode maximum (remarquez la présence du 8288).



Note : inutile en mode 8088

Fig. 11. — Connexions à réaliser pour adapter un 8087 à un 8086 (8088) en mode maximum.



Note : inutile en mode 8088

Fig. 11. — Connexions à réaliser pour adapter un 8087 à un 8086 (8088) en mode maximum.

Vous pouvez préférer adapter un 8088 sur votre carte SDK 86 — ou autre —, vous trouverez le schéma du dispositif figure 12.

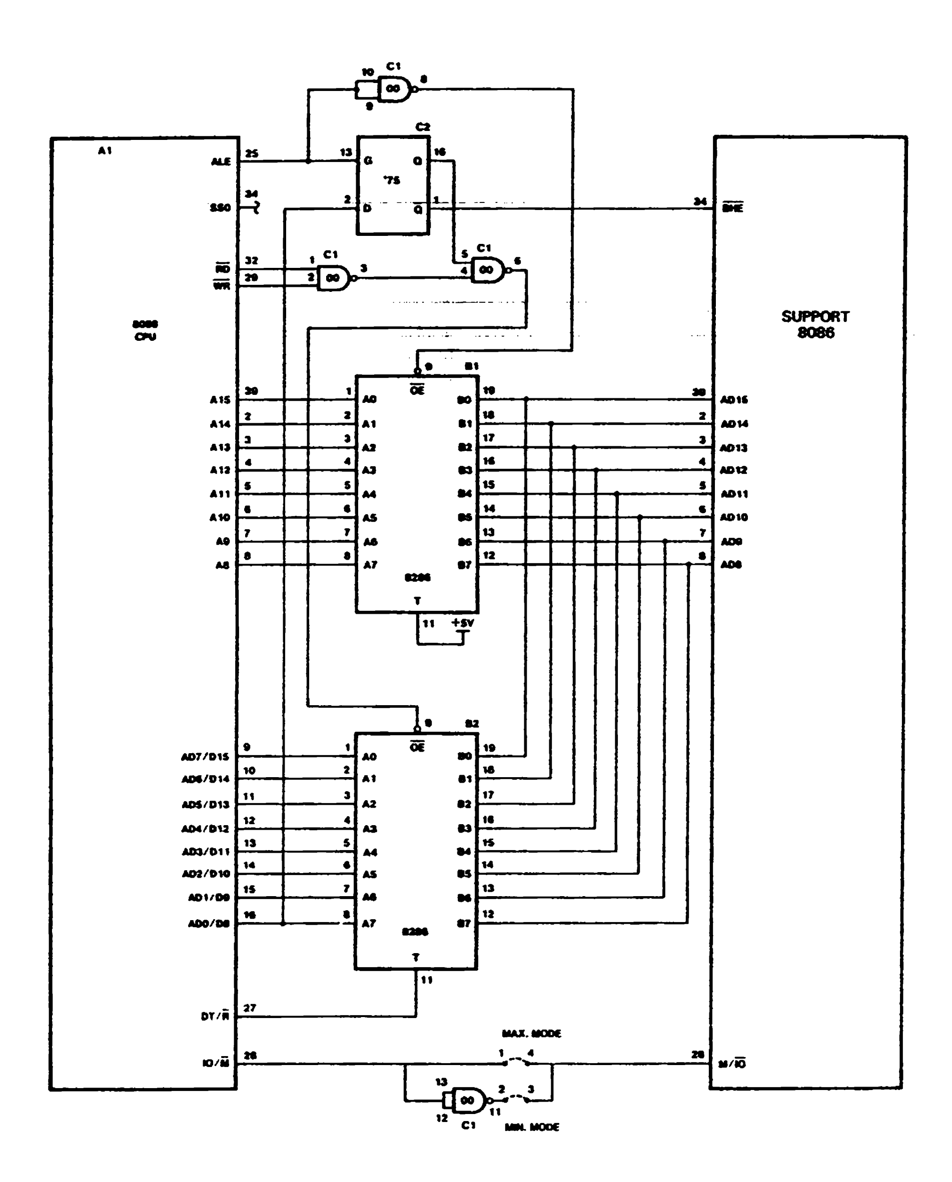


Fig. 12. — Schéma d'adaptation d'un 8088 sur support 8086.

Les figures 13 et 14 vous indiquent comment connecter, en plus du 8087, un ou deux processeurs entrée/sortie 8089... attention à la programmation en codes machine, le seul mode permis avec le SDK 86, à moins de connecter cette carte à un outil de développement ou de disposer d'un moniteur plus performant !

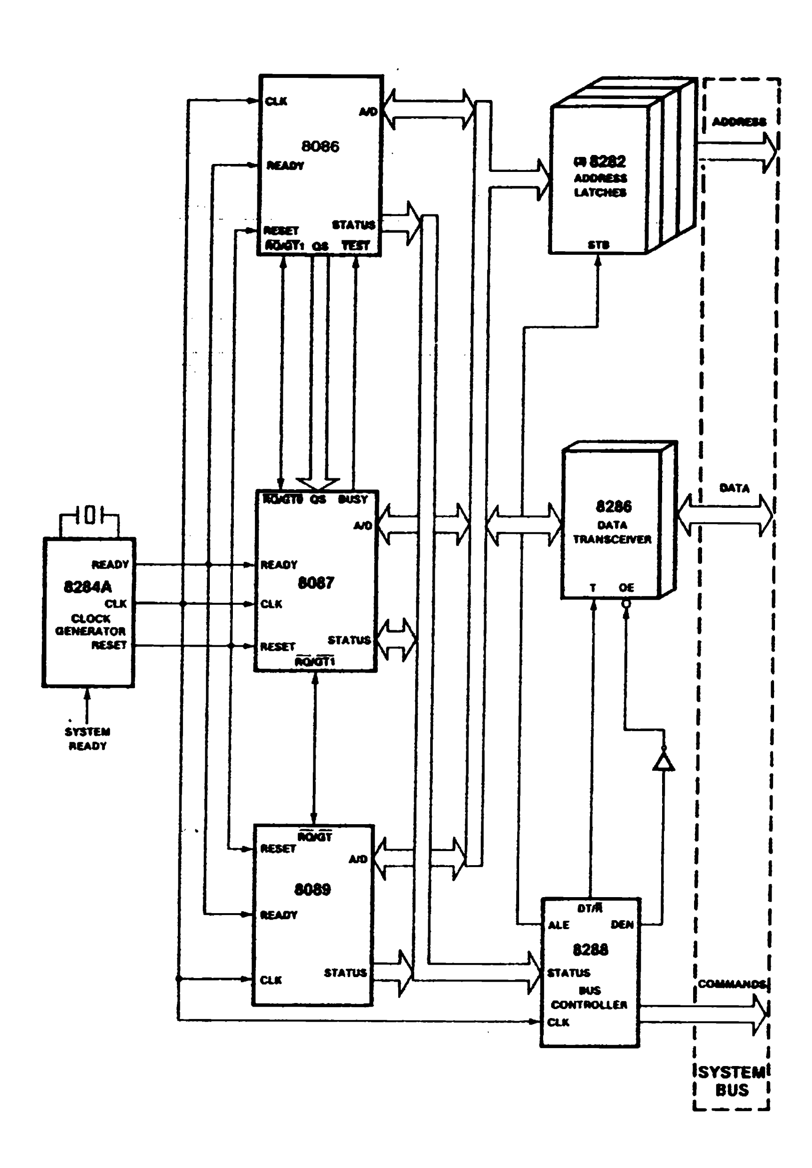


Fig. 13. — Schéma de câblage comportant le 8086 associé au coprocesseur arithmétique 8087 et au processeur d'entrée/sortie 8089.

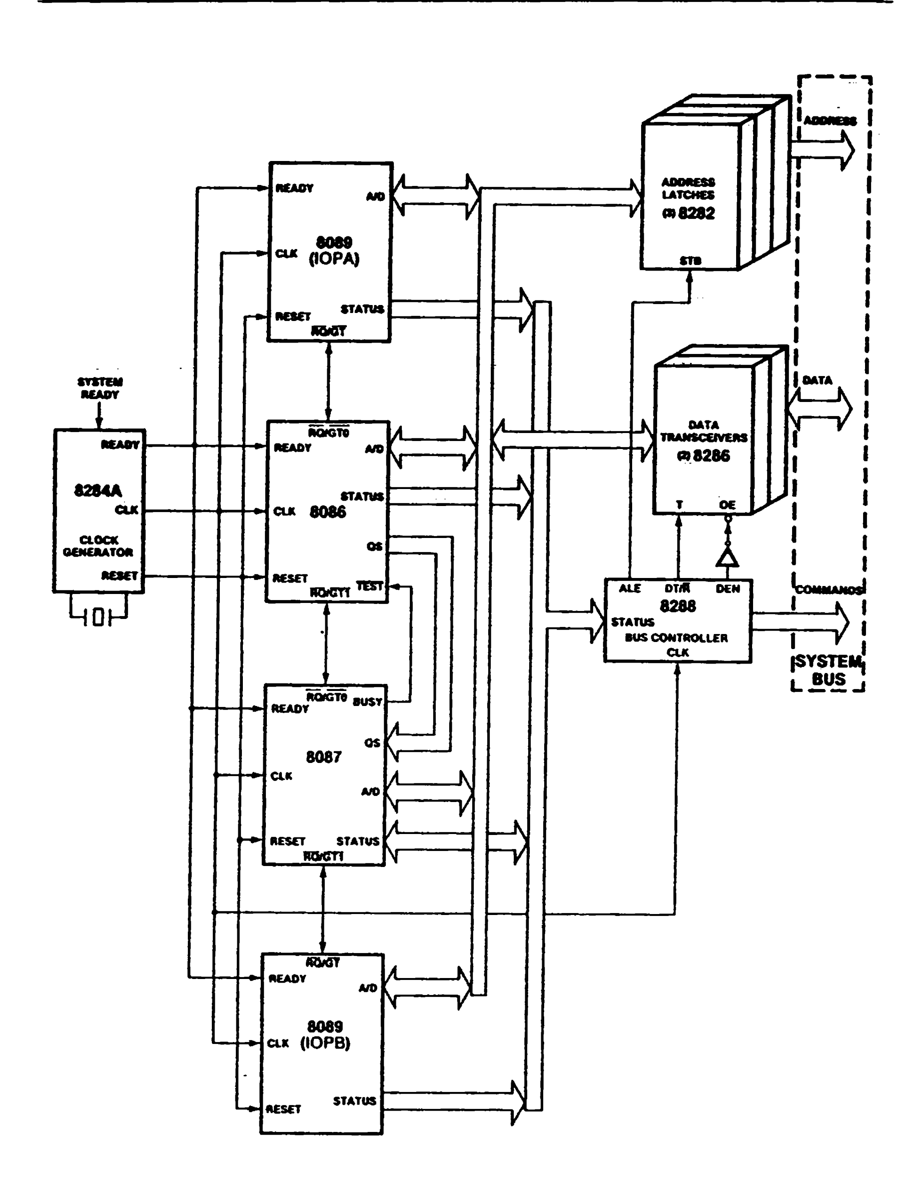


Fig. 14. — 8086 associé à un 8087 et deux 8089.

TABLEAU DES INSTRUCTIONS ET DE LEURS DUREES D'EXECUTION

— Pour 8086 et 8088, il faut tenir compte de la durée du calcul de Adresse Effective (AE) donnée dans le tableau 9.

— Pour 8086, si le mot concerné est à une adresse impaire, il faut ajouter 4 périodes.

— Pour 8088 et iAPX 188, les durées indiquées sont celles concernant les mots de 16 bits, celles concernant les mots de 8 bits étant identiques à celles du 8086 ou iAPX 186 (sauf certaines opérations ne traitant que des mots de 16 bits PUSH, POP...).

— Pour iAPX 186 et 188, les mots sont à des adresses paires, il faut pour certains calculs de AE ajouter 1 (parfois 2) périodes.

— Pour iAPX 286, les données ne concernent que le travail en mode réel (1 M octets), toutefois les instructions utilisées en mode protégé ou virtuel sont indiquées. Il faut ajouter une période pour un calcul de AE nécessitant trois termes. Pour les sauts, appels et retours « m » est le nombre d'octets de l'instruction suivante.

INSTRUCTIONS DES iAPX 86, 88, 186, 286

TRANSFERT DE DONNEES

FONCTION	FORMAT				DUREE (mots de 16 bits)				
					86	88	186	188	286
MOV									
Registre vers registre/mémoire	1000 100 w	mod reg r/m			2/9+AE	2/13+AE	2/12	2/16	2/3
Registre/mémoire vers registre	1000 101 w	mod reg r/m			2/8+AE	2/12+AE	2/9	2/13	2/5
Immédiat vers registre/mémoire	1100 011 w	mod 000 r/m	donnée	donnée si w=1	4/10+AE	4/14+AE	12-13	12-17	2/3
Immédiat vers registre	1011 w reg	donnée	donnée si w=1		4	4	3-4	3-4	2
Mémoire vers accumulateur	1010 000 w	adresse basse	adresse haute		10	14	9	13	5
Accumulateur vers mémoire	1010 001 w	adresse basse	adresse haute		10	14	8	12	3
Registre/mémoire vers segment	1000 1110	mod 0seg r/m	seg≠01 (CS)		2/8+AE	2/12+AE	2/9	2/13	2/5
Segment vers registre/mémoire	1000 1100	mod 0seg r/m			2/9+AE	2/13+AE	2/11	2/15	2/3
PUSH									
Mémoire	1111 1111	mod 110 r/m			16+AE	24+AE	16	20	5
Registre	010 10reg				11	15	10	14	3

Segment	000 seg 110 (CS légal)	10	14	9	13	3
Immédiat	0110 10s0 / donnée / donnée si s=0	—	—	10	14	3
PUSHA (PUSH All)	0110 0000	—	—	36	68	17
PUSHF (PUSH Flags)	1001 1100	10	14	9	13	3
POP						
Mémoire	1000 1111 / mod 000 r/m	17+AE	25+AE	20	24	5
Registre	0101 1reg	8	12	10	14	5
Segment	000 seg 111 (CS illégal : seg≠01)	8	12	8	12	20
POPA (POP All)	0110 0001	—	—	51	83	19
POPF (POP Flags)	1001 1101	8	12	8	12	5
XCHG (Echange)						
Registre avec registre/mémoire	1000 011w / mod reg r/m	4/17+AE	4/25+AE	4/17	4/25	3/5
Registre avec accumulateur	100 10reg	3	3	3	3	3
IN (Entrée depuis un)						
Port défini	1110 010w / port	10	14	10	14	5
Port variable (DX)	1110 110w	8	12	8	12	5
OUT (Sortie vers un)						
Port défini	1110 011w / port	10	14	9	13	3

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

TRANSFERT DE DONNEES (suite)

FONCTION	FORMAT			DUREE (mots de 16 bits)				
				86	88	186	188	286
Port variable (DX)	1110 111w			8	12	7	11	3
XLAT Traduction vers AL	1101 0111			11	11	11	15	5
LEA Charge AE dans un registre	1000 1101	mod reg r/m	(mod ≠ 11)	2+AE	2+AE	6	6	3
LDS Charge registre et DS	1100 0101	mod reg r/m	(mod ≠ 11)	16+AE	24+AE	18	26	21
LES Charge registre et ES	1100 0100	mod reg r/m	(mod ≠ 11)	16+AE	24+AE	18	26	21
LAHF Charge flag dans AX	1001 1111			4	4	2	2	2
SAHF Charge flag avec AX	1001 1110			4	4	3	2	

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

ARITHMETIQUE

FONCTION	FORMAT				DUREE (mots de 16 bits)				
					86	88	186	188	286
ADD									
Registre + registre	0000 00dw	mod reg r/m			3	3	3	3	2
Registre + mémoire					9+AE	13+AE	10	14	7
Mémoire + registre					16+AE	24+AE	10	18	7
Registre/mémoire + donnée	1000 00sw	mod 000 r/m	donnée	donnée si sw=01	4/17+AE	4/25+AE	4/16	4/24	3/7
Accumulateur + donnée	0000 010w	donnée	donnée si w=1		4	4	3-4	3-4	3
ADC (ADD + Carry)									
Registre + registre + CF	0001 00dw	mod reg r/m			3	3	3	3	2
Registre + mémoire + CF					9+AE	13+AE	10	14	7
Mémoire + registre + CF					16+AE	24+AE	10	18	7
Registre/mémoire + donnée + CF	1000 00sw	mod 010 r/m	donnée	donnée si sw=01	4/17+AE	4/25+AE	4/16	4/24	3/7
Accumulateur + donnée + CF	0001 010w	donnée	donnée si w=1		4	4	3-4	3-4	3
INC (Incrément)									
Registre/mémoire	1111 111w	mod 000 r/m			3/15+AE	3/23+AE	3/15	3/23	2/7
Registre (16 bits)	0100 0reg				2	2	3	3	2

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

ARITHMETIQUE (suite)

FONCTION	FORMAT				DUREE (mots de 16 bits)				
					86	88	186	188	286
SUB									
Registre-registre	0010 10dw	mod reg r/m			3	3	3	3	2
Registre-mémoire					9+AE	13+AE	10	14	7
Mémoire-registre					16+AE	24+AE	10	18	7
Registre/mémoire-donnée	1000 00sw	mod 101 r/m	donnée	donnée si sw=01	4/17+AE	4/25+AE	4/16	4/24	3/7
Accumulateur-donnée	0010 110w	donnée	donnée si w=1		4	4	3-4	3-4	3
SBB (SUB-Carry)									
Registre-registre-CF	0001 10dw	mod reg r/m			3	3	3	3	2
Registre-mémoire-CF					9+AE	13+AE	10	14	7
Mémoire-registre-CF					16+AE	24+AE	10	18	7
Registre/mémoire-donnée-CF	1000 00sw	mod 011 r/m	donnée	donnée si sw=01	4/17+AE	4/25+AE	4/16	4/24	3/7
Accumulateur-donnée-CF	0001 110w	donnée	donnée si w=1		4	4	3-4	3-4	3
DEC (Decrement)									
Registre-mémoire	1111 111w	mod 001 r/m			3/15+AE	3/23+AE	3/15	3/23	2/7
Registre (16 bits)	0100 1reg				2	2	3	2	2

CMP (Comparaison)									
Registre avec registre/mémoire	0011 100w	mod reg r/m			3/9+AE	3/13+AE	3/10	3/14	2/6
Registre/mémoire avec registre	0011 101w	mod reg r/m			3/9+AE	3/13+AE	3/10	3/14	2/7
Registre/mémoire avec donnée	1000 00sw	mod 111 r/m	donnée	donnée si sw=01	4/10+AE	4/14+AE	3/10	3/14	3/6
Accumulateur avec donnée	0011 110w	donnée	donnée si w=1		4	4	3-4	3-4	3
NEG (Changement de signe)	1111 011w	mod 011 r/m			3/16+AE	3/24+AE	3	3	2
AAA (Ajustement ASCII pour +)	0011 0111				4	4	8	8	3
DAA (Ajustement Dec. pour +)	0010 0111				4	4	4	4	3
AAS (Ajustement ASCII pour −)	0011 1111				8	8	7	7	3
DAS (Ajustement Dec. pour −)	0010 1111				4	4	4	4	3
MUL (non signée)	1111 011w	mod 100 r/m							
Registre 8 bits					70-77	70-77	26-28	26-28	13
Registre 16 bits					118-133	118-133	35-37	35-37	21
Mémoire 8 bits					(76-83)+AE	(76-83)+AE	32-34	32-34	16
Mémoire 16 bits					(124-139)+AE	(128-143)+AE	41-43	45-47	24
IMUL (signée)	1111 011w	mod 101 r/m							
Registre 8 bits					80-98	80-98	25-28	25-28	13
Registre 16 bits					128-154	128-154	34-37	34-37	21
Mémoire 8 bits					(86-104)+AE	(86-104)+AE	31-34	31-34	16
Mémoire 16 bits					(134-160)+AE	(138-164)+AE	40-43	44-47	24
IMUL immédiat (signée)	0110 10s1	mod reg r/m	donnée	donnée si s=0	−	−	22-25/29-32	22-25/33-36	21/24

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

ARITHMETIQUE (suite)

FONCTION	FORMAT		DUREE (mots de 16 bits)				
			86	88	186	188	286
DIV (non signée)	1111 011w	mod 110 r/m					
Registre 8 bits			80-90	80-90	29	29	14
Registre 16 bits			144-162	144-162	38	38	22
Mémoire 8 bits			(86-96)+AE	(86-96)+AE	35	35	17
Mémoire 16 bits			(150-168)+AE	(154-172)+AE	44	48	25
IDIV (signée)	1111 011w	mod 111 r/m					
Registre 8 bits			101-112	101-112	44-52	44-52	17
Registre 16 bits			165-184	165-184	53-61	53-61	25
Mémoire 8 bits			(107-118)+AE	(107-118)+AE	50-58	50-58	20
Mémoire 16 bits			(134-160)+AE	(138-164)+AE	59-67	63-71	28
AAM (Ajustement ASCII pour multipl.)	1101 0100	0000 1010	83	83	19	19	16
AAD (Ajustement ASCII pour division)	1101 0101	0000 1010	60	60	15	15	14
CBW 8 bits 16 bits	1001 1001		2	2	2	2	2
CWD 16 bits 32 bits	1001 1001		5	5	4	4	2

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

LOGIQUE

FONCTION	FORMAT	DUREE (mots de 16 bits) 86	88	186	188	286
Décalages et rotations						
Registre 1 fois	1101 000w mod TTT r/m	2	2	2	2	2
Mémoire 1 fois		15+AE	23+AE	15	23	7
Registre (CL) fois	1101 001w mod TTT r/m	8+4.n	8+4.n	5+n	5+n	5+n
Mémoire (CL) fois		20+AE+4.n	28+AE+4.n	17+n	25+n	8+n
Registre n fois	1100 000w mod TTT r/m n (n ⩽ 31)	—	—	5+n	5+n	5+n
Mémoire n fois				17+n	25+n	8+n
	TTT 000 ROL 001 ROR 010 RCL 011 RCR 100 SAL/SHL 101 SHR 111 SAR					
AND (ET)						
Registre ET registre	0010 00dw mod reg r/m	3	3	3	3	3
Registre ET mémoire		9+AE	13+AE	10	14	7
Mémoire ET registre		16+AE	24+AE	10	18	7
Registre ET donnée	1000 000w mod 100 r/m donnée donnée si w=1	4	4	4	4	3
Mémoire ET donnée		17+AE	25+AE	16	24	7
Accumulateur ET donnée	1010 100w donnée donnée si w=1	4	4	3-4	3-4	3

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

LOGIQUE

FONCTION	FORMAT				DUREE (mots de 16 bits)				
					86	88	186	188	286
TEST (ET, flags seuls affectés)									
Registre/mémoire et registre	1000 010w	mod reg r/m			3/9+AE	3/13+AE	3/10	3/14	2/6
Registre/mémoire et donnée	1111 011w	mod 000 r/m	donnée	donnée si w=1	5/11+AE	5/19+AE	4/10	4/18	3/6
Accumulateur et donnée	1010 100w	donnée	donnée si w=1		4	4	3-4	3-4	3
OR (OU)									
Registre OU registre/mémoire Mémoire OU registre	0000 10dw	mod seg r/m			3/9+AE 16+AE	3/13+AE 24+AE	3/10	3/14	2/7
Registre/mémoire OU donnée	1000 00dw	mod 001 r/m	donnée	donnée si w=1	4/17+AE	4/25+AE	4/16	4/24	3/7
Accumulateur OU donnée	0000 110w	donnée	donnée si w=1		4	4	3-4	3-4	3
XOR (OU exclusif : +)									
Registre + registre/mémoire Mémoire + registre	0011 00dw	mod reg r/m			3/9+AE 16+AE	3/13+AE 24+AE	3/10	3/14	2/7
Registre/mémoire + donnée	1000 000w	mod 110 r/m	donnée	donnée si w=1	4/17+AE	4/25+AE	4/16	4/24	3/7
Accumulateur + donnée	0011 010w	donnée	donnée si w=1		4	4	3-4	3-4	3
NOT (complément à 1)	1111 011w	mod 010 r/m			3/16+AE	3/24+AE	3/10	3	2/7

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

MANIPULATION DE SUITES

Simple

FONCTION	FORMAT	DUREE (mots de 16 bits)				
		86	88	186	188	286
MOVS Transfert de bloc	1010 010w	18	26	14	22	5
CMPS Comparaison	1010 011w	22	30	22	30	8
SCAS « Balayage »	1010 111w	15	19	15	19	7
LODS Chargement de AL/AX	1010 110w	12	16	12	16	5
STOS Stockage de AL/AX	1010 101w	11	15	10	14	3
INS Entrée	0110 110w	–	–	14	18	5
OUTS Sortie	0110 111w	–	–	14	18	5

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

MANIPULATION DE SUITES

Répétée

FONCTION	FORMAT		DUREE (mots de 16 bits)				
			86	88	186	188	286
REP MOVS Transfert du bloc	1111 0011	1010 010w	9+17*n	9+25*n	8+8*n	8+16*n	5+4*n
REPZ/REPNZ CMPS Comparaison	1111 001z	1010 011w	9+22*n	9+30*n	5+22*n	5+30*n	5+9*n
REPZ/REPNZ SCAS « Balayage »	1111 001z	1010 111w	9+15*n	9+19*n	5+15*n	5+19*n	5+8*n
REP LODS Chargement de AL/AX	1111 0011	1010 110w	9+13*n	9+17*n	6+11*n	5+15*n	5+4*n
REP STOS Stockage de AL/AX	1111 0011	1010 101w	9+10*n	9+14*n	6+9*n	6+13*n	4+3*n
REP INS Entrée	1111 0011	0110 110w	–	–	8+8*n	8+12*n	5+4*n
REP OUTS Sortie	1111 0011	0110 111w	–	–	8+8*n	8+12*n	5+4*n

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

TRANSFERT (PROGRAMME)

FONCTION	FORMAT	DUREE (mots de 16 bits) 86	88	186	188	286
CALL (Appel de procédure)						
Direct intrasegment	1110 1000 · DEC. BAS · DEC. HAUT	19	23	14	18	7+m
Indirect (registre/mémoire) intrasegment	1111 1111 · mod 010 r/m	16/21+AE	24/29+AE	13/19	17/27	7+m/11+m
Direct intersegment	1001 1010 · nouveau IP ·	28	36	23	31	13+m
	nouveau CS ·					
Indirect intersegment	1111 1111 · mod 011 r/m · mod≠11	37+AE	57+AE	38	54	16+m
JMP (Saut inconditionné)						
Court	1110 1011 · DEC	15	15	13	13	7+m
Long intrasegment	1110 1001 · DEC. BAS · DEC. HAUT	15	15	13	13	7+m
Court indirect (registre/mémoire) intrasegment	1111 1111 · mod 100 r/m	11/18+AE	11/18+AE	11/17	11/21	7+m/11+m
Direct intersegment	1110 1010 · nouveau IP ·	15	15	13	13	11+m
	nouveau CS ·					
Indirect intersegment	1111 1111 · mod 101 r/m · mod≠11	24+AE	24+AE	26	34	15+m

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

TRANSFERT (PROGRAMME)
(suite)

FONCTION	FORMAT			DUREE (mots de 16 bits)				
				86	88	186	188	286
RET (Retour)								
Intrasegment	1100 0011			8	20	16	20	11+m
Intrasegment avec décalage (SP)	1100 0010	DEC. BAS	DEC. HAUT	12	24	18	22	11+m
Intersegment	1100 1011			18	32	22	30	15+m
Intersegment avec décalage (SP)	1100 1010	DEC. BAS	DEC. HAUT	17	31	25	33	15+m
SAUTS Conditionnels								
JE/JZ Egal ou nul	0111 0100	DEC.		4/16	4/16	4/13	4/13	3/7+m
JL/JNGE Plus petit que	0111 1100	DEC.		"	"	"	"	"
JLE/JNG Plus petit que ou égal à	0111 1110	DEC.		"	"	"	"	"
JB/JNAE Inférieur à	0111 0010	DEC.		"	"	"	"	"
JBE/JNA Inférieur ou égal à	0111 0110	DEC		"	"	"	"	"
JP/JPE Parité paire	0111 1010	DEC.		"	"	"	"	"

JO Overflow	0111 0000	DEC	"	"	"	"	"
JC Carry	0111 0010	DEC	"	"	"	"	"
JNC Pas de carry	0111 0011	DEC	"	"	"	"	"
JS Négatif	0111 1000	DEC.	"	"	"	"	"
JNE/JNZ Différent de	0111 0101	DEC.	"	"	"	"	"
JNL/JGE Plus grand que ou égal à	0111 1101	DEC.	"	"	"	"	"
JNLE/JG Plus grand que	0111 1111	DEC.	"	"	"	"	"
JNB/JAE Supérieur ou égal à	0111 0011	DEC.	"	"	"	"	"
JNBE/JA Supérieur à	0111 0111	DEC.	"	"	"	"	"
JNP/JPO Parité impaire	0111 1011	DEC.	"	"	"	"	"
JNO Pas d'overflow	0111 0001	DEC.	"	"	"	"	"
JNS Positif	0111 1001	DEC.	"	"	"	"	"
LOOP Boucle (CX) fois	1110 0010	DEC.	5/17	5/17	5/15	5/15	4/8+m
LOOPZ/LOOPE Boucle tant que égal ou zéro	1110 0001	DEC.	8/16	8/16	6/16	6/16	4/8+m
LOOPNZ/LOOPNE Boucle tant que différent de	1110 0000	DEC.	8/16	8/16	6/16	6/16	4/8+m
JCXZ Si (CX)=0	1110 0011	DEC.	8/16	8/16	5/16	6/16	4/8+m

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

INTERRUPTIONS

FONCTION	FORMAT	DUREE (mots de 16 bits)				
		86	88	186	188	286
INT						
Type précisé	1100 1101 | TYPE	52	71	47	47	23+m
Type 3	1100 1100	51	72	45	45	23+m
INTO Overflow	1100 1110	4/53	4/73	4/48	4/48	3/23+m
IRET Retour	1100 1111	24	44	28	28	17+m

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

CONTROLE DU PROCESSEUR

FONCTION	FORMAT	DUREE (mots de 16 bits)				
		86	88	186	188	286
CLC $\overline{Carry}$=0	1111 1000	2	2	2	2	2
CMC $\overline{Carry}$=$\overline{Carry}$	1111 0101	2	2	2	2	2
STC $\overline{Carry}$=1	1111 1001	2	2	2	2	2
CLD DF=0	1111 1100	2	2	2	2	2
STD DF=1	1111 1101	2	2	2	2	2
CLI IF=0	1111 1010	2	2	2	2	2
STI IF=1	1111 1011	2	2	2	2	2
HLT Halte	1111 0100	2	2	2	2	2
WAIT Attente	1001 1011 si $\overline{test}$=0	3+5n	3+5n	6	6	3
LOCK Verrouillage (préfixe)	1111 0000	2	2	2	2	0

INSTRUCTIONS DES iAPX 86, 88, 186, 188, 286

CONTROLE DU PROCESSEUR
(suite)

FONCTION	FORMAT	DUREE (mots de 16 bits)				
		86	88	186	188	286
NOP Pas d'opération	1001 0000	3	3	3	3	3
ESC Instruction pour extension	1101 1TTT | mod LLL r/m (TTTLLL = code opération pour l'extension)	2/8+AE	2/12+AE	6	6	9/20

INSTRUCTIONS N'EXISTANT PAS POUR 8086/8088

FONCTION	FORMAT	DUREE (mots de 16 bits)		
		186	188	286
ENTER Entrée d'une procédure L = 0 L = 1 (L = *Level*, Niveau) L > 1	1100 100v | donnée basse | donnée haute | L	 15 25 22+16(L-1)	 15 25 22+16(L-8)	 11 11 18+4(L-1)
LEAVE Sortie d'une procédure	1100 1001	8	8	5
BOUND Détection d'une sortie de limites	0110 0010 | mod reg r/m	33/35	33/35	13

INSTRUCTIONS PROPRES A iAPX 286

FONCTION	FORMAT				DUREE (mots de 16 bits)
					286
CLTS Flag de tâche à zéro	0000 1111	0000 0110			2
LGDT Charger la table globale	0000 1111	0000 0001	mod 010 r/m	mod+11	11
SGDT Stocker la table globale	0000 1111	0000 0001	mod 000 r/m	mod+11	11
LIDT Charger la table d'interruption	0000 1111	0000 0001	mod 011 r/m	mod+11	12
SIDT Stocker la table d'interruption	0000 1111	0000 0001	mod 001 r/m	mod+11	12
LLDT Charger le registre de la table locale	0000 1111	0000 0000	mod 010 r/m		Mode protégé
SLDT Stocker le registre de la table locale	0000 1111	0000 0000	mod 000 r/m		
LTR Charger le registre de tâche	0000 1111	0000 0000	mod 011 r/m		
STR Stocker le registre de tâche	0000 1111	0000 0000	mod 001 r/m		
LMSW Charger le mot d'état machine	0000 1111	0000 0001	mod 110 r/m		3/6
SMSW Stocker le mot d'état machine	0000 1111	0000 0001	mod 100 r/m		2/3

INSTRUCTIONS PROPRES A iAPX 286
(suite)

FONCTION	FORMAT			DUREE (mots de 16 bits)	
					286
LAR Charger le registre des règles d'accès	0000 1111	0000 0010	mod reg r/m		Mode protégé
LSL Charger le registre de limite de segment	0000 1111	0000 0010	mod reg r/m		
ARPL Ajuster le niveau de privilège	0110 0011	mod reg r/m			
VERR Vérifier si un segment peut être lu	0000 1111	0000 0000	mod 100 r/m		
VERW Vérifier si un segment peut être écrit	0000 1111	0000 0000	mod 101 r/m		

MATRICE DES CODES-MACHINES

Haut \ Bas	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ADD b.f.r/m	ADD w.f.r/m	ADD b.t.r/m	ADD w.t.r/m	ADD b.ia	ADD w.ia	PUSH ES	POP ES	OR b.f.r/m	OR w.f.r/m	OR b.t.r/m	OR w.t.r/m	OR b.ia	OR w.ia	PUSH CS	[illegible]
1	ADC b.f.r/m	ADC w.f.r/m	ADC b.t.r/m	ADC w.t.r/m	ADC b.ia	ADC w.ia	PUSH SS	POP SS	SBB b.f.r/m	SBB w.f.r/m	SBB b.t.r/m	SBB w.t.r/m	SBB b.ia	SBB w.ia	PUSH DS	POP DS
2	AND b.f.r/m	AND w.f.r/m	AND b.t.r/m	AND w.t.r/m	AND b.ia	AND w.ia	SEG -ES	DAA	SUB b.f.r/m	SUB w.f.r/m	SUB b.t.r/m	SUB w.t.r/m	SUB b.ia	SUB w.ia	SEG -CS	DAS
3	XOR b.f.r/m	XOR w.f.r/m	XOR b.t.r/m	XOR w.t.r/m	XOR b.ia	XOR w.ia	SEG -SS	AAA	CMP b.f.r/m	CMP w.f.r/m	CMP b.t.r/m	CMP w.t.r/m	CMP b.ia	CMP w.ia	SEG -DS	AAS
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6	PUSHA	POPA	BOUND						PUSH b.i.	IMUL b.r/m	PUSH w.i.	IMUL w.r/m	INS b	INS w	OUTS b	OUTS w
7	JO	JNO	JC/JB/ JNAE	JNC/JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG
8	[illegible]	[illegible]	[illegible]	[illegible]	TEST b.r/m	TEST w.r/m	XCHG b.r/m	XCHG w.r/m	MOV b.f.r/m	MOV w.f.r/m	MOV b.t.r/m	MOV w.t.r/m	MOV sr.f.r/m	LEA r/m	MOV sr.t.r/m	POP r/m
9	XCHG AX	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI	CBW	CWD	CALL l.d	WAIT	PUSHF	POPF	SAHF	LAHF
A	MOV m - AL	MOV m - AX	MOV AL - m	MOV AX m	MOVS b	MOVS w	CMPS b	CMPS w	TEST b.ia	TEST w.ia	STOS b	STOS w	LODS b	LODS w	SCAS b	SCAS w
B	MOV i - AL	MOV i - CL	MOV i - DL	MOV i - BL	MOV i - AH	MOV i - CH	MOV i - DH	MOV i - BH	MOV i - AX	MOV i - CX	MOV i - DX	MOV i - BX	MOV i - SP	MOV i - BP	MOV i - SI	MOV i - DI
C	[illegible]	[illegible]	RET. c(i+SP)	RET c	LES r/m	LDS r/m	MOV b.i.r/m	MOV w.i.r/m	ENTER	LEAVE	RET. l.(i+SP)	RET l	INT Type 3	INT (Tous)	INTO	IRET
D	[illegible]	[illegible]	[illegible]	[illegible]	AAM	AAD		XLAT	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCXZ	IN b	IN w	OUT b	OUT w	CALL d	JMP d	JMP l.d	JMP c,d	IN v,b	IN v,w	OUT v,b	OUT v,w
F	LOCK		REP REPNZ	REPZ	HLT	CMC	[illegible]	[illegible]	CLC	STC	CLI	STI	CLD	STD	[illegible]	[illegible]

où :

mod ☐ r/m	000	001	010	011	100	101	110	111
Immed	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL/SAL	SHR	—	SAR
Grp1	TEST	—	NOT	NEG	MUL	IMUL	DIV	IDIV
Grp2	INC	DEC	CALL id	CALL l.id	JMP id	JMP l.id	PUSH	—

Mode prot. instruction pour iAPX 286 en mode protégé

b = mot de 8 bits
d = direct
f = depuis un registre
i = immédiat
ia = immédiat vers accu.
id = indirect
is = immediat 8 bits étendus à 16 bits
l = saut/appel intersegment

m = mémoire
r/m = le 2ᵉ octet donne le mode d'adressage
c = saut/appel intrasegment
sr = registre de segment
t = vers un registre
v = variable
w = mot de 16 bits

BIBLIOGRAPHIE

The 8086 PRIMER	— Stephen P. Morse/Hayden
The 8086 Book incluced the 8088	— Russel Rector/George Alexy/ A. Osborne/Mac Grawhill
iAPX 186, 188 User's Manuel	— INTEL
iAPX 286 Programmer's Reference Manual	— INTEL
80386 Advance Information	— INTEL (avril 1986)
Microsystem Components Handbook 1986	— INTEL
Notes d'applications AP 67, AP 76 et AP 113	— INTEL

TABLE DES MATIERES

H

I

J

L

P

R

S